PROMPT 48 MICROCOMPUTER USER'S MANUAL

Manual Order Number: 9800402C

Copyright © 1976, 1977, 1978 Intel Corporation
Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

The information in this manual is subject to change without notice. Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this manual. Intel Corporation makes no commitment to update nor to keep current the information contained in this manual.

No part of this manual may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation. The following are trademarks of Intel Corporation and may be used only to describe Intel products:

ICE	LIBRARY MANAGER	PROMPT
INSITE	MCS	RMX
INTEL	MEGACHASSIS	UPI
INTELLEC	MICROMAP	μSCOPE
iSBC	MULTIBUS	

Printed in U.S.A./B-50/0578/4K CP

PREFACE

This User's Manual contains the information you will need to use your PROMPT 48. The information presented herein is adequate to support normal user needs. Additional information is available in the following documents.

MCS-48 Microcomputer User's Manual, Order No. 9800270

MSC-48 Assembly Language Manual, Order No. 9800255

PROMPT 48 Reference Cardlet, Order No. 9800404

CONTENTS

CONTENTS (Continued)

TABLES

ILLUSTRATIONS

CHAPTER 1
INTRODUCTION

1-1. How To Use This Book

The cost of computers is now low enough that your software design and debug time is likely to be a critical consideration. No doubt your decision to use good tools like Prompt 48 was based on this kind of logical thinking. Since your time is valuable, this book is organized as a reference work, not as a mystery story. Every page has headings that identify the topics on that page. Look up what you want to know, in whatever order you need the information. If Prompt 48 is new to you, you probably will want to go through the familiarization exercise in Appendix A. Before operating Prompt 48 for the first time, please check the caution items that follow.

1-2. Voltage Selection

Is the voltage selection switch on the back of Prompt 48 set for your local mains (line) voltage? If not, open the Prompt box, remove the switch locking plate, and set the switch properly, then reassemble the unit. If you change the switch setting, the fuse likely must be changed to correspond. Ratings are:

105–125 V – 2 A
208–250 V – 1 A

Now you may plug Prompt 48 in and turn it on.

1-3. Handling The Processor

THE CHIP COMPUTER IS FRAGILE! Dropping, twisting, or uneven pressure may break it. Leave it in its protective package until ready to use it. Never press down upon the quartz window area of the processor, or exert twisting or bending forces on any device. Never subject any MOS device to the discharge of static electricity; touch the chassis of Prompt 48 before inserting a device in the socket on its panel.

1-4. Inserting Processor In Execution Socket

CAUTION

Never insert a processor in the PROGRAMMING SOCKET unless a second processor is properly locked in the EXECUTION SOCKET.

Release the locking lever. Gently seat the processor in the Execution Socket, notched end away from you. Move the locking lever down flush with the panel.

1-5. The Purpose of Prompt 48

The difference between a computer and other calculating or controlling devices is the general-purpose nature of their programmability. The 8048 is a true general-purpose digital computer. Its purpose is undetermined until you design software for it, commit that software design to a mask, and maunfacture the chip.

Prompt 48 is a tool to aid you in learning MCS-48 programming and in writing, debugging, and testing the programs you write. There is enough information here to get you started, whether or not you have ever written a program before.

Prompt 48 is a machine-language computer; making it support assembly-language programming would have considerably raised its cost. Even so, it is general purpose, and can be used to perform a variety of tasks, among which are the control of TTL-compatible devices and the programming of PROMs. It can function as an Intellec Microcomputer Development System peripheral in the latter respect. Once a program has been deposited in an 8748 computer, that device can be installed in the EXECUTION SOCKET on the panel of Prompt 48. The pins of either executory processor—8748 or 8035—can be directly interfaced to your prototype via the I/O PORTS AND BUS CONNECTOR and a cable set provided with Prompt.

All of Prompt 48's circuitry is located on a single board just beneath the panel. Aside from the power supply, the remainder of the Prompt 48's cabinet is empty. A slot at the back of the cabinet provides access for interconnections.

1-6. Getting Started

Entering a program into Prompt 48's random-access memory (RAM) is easy. The example that follows can be loaded and run without any more instructions than are given here in this paragraph. (The MCS-48 Assembly Language Manual has some other sample programs of a tutorial nature.) Do the following, step by step, and you will be running a program in a matter of minutes.

a. Connect power to Prompt 48.

b. Install the 8035 computer in the EXECUTION SOCKET. (Observe the precautions in paragraph 1-3.)

c. Turn power ON. The display should respond with ACCESS = 0. If not, press [SYS RST].

d. Enter the program by pressing each COMMANDS or HEX DATA/FUNCTIONS key in the order listed on the next page. Each [ ] represents one keystroke. At the end of each step (which may be several keystrokes), the results shown in the column at right should appear on the display. If you make a mistake and the wrong data appears, you can correct it by keying the field over again before touching the NEXT [,] key. If you realize a mistake after incrementing to the next address, you can go back and correct it by pressing the [ ] CLEAR ENTRY/PREVIOUS key and then keying the step over again.

Step	Action	Function	Result Address	Data	Instruction Mnemonic	Comment
1.	[ ] EXAMINE/MODIFY	E				;SELECT FUNCTION
	[ ] PROGRAM MEMORY	EP				;SELECT PROGRAM MEMORY
	[0]	EP	0			;ADDRESS 0
2.	[,] [1] [7]	EP	0	17	INC A	;INCREMENT ACCUMULATOR
3.	[,] [0] [4]	EP	1	04	JMP	;JUMP TO LOCATION
	[,] [0]	EP	2	00		;00

CHAPTER 2
THE NUMBER SYSTEM AND ITS SYMBOLS

2-1. Why Computers Need Symbols

Digital computers perform functions accurately and at high speed by manipulating symbols (characters) according to a set of instructions. Computer operation consists of the execution of sequences of symbolically coded instructions and data. Within the machine, both data and instructions are usually described in binary-number codes.

To understand the computer, you will need to understand how numbers are represented. Our starting point is the study of the simplest of numbering systems—the binary number system. But first, some definitions.

2-2. Number Systems

A number system is a set of symbols that may be operated upon by arithmetic rules. The individual symbols are called digits, and each digit is assigned its own name. The decimal system, as the name suggests, has ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A number system also has a set of rules that define how to arrange the digits to form numbers. A number is, therefore, a sequence of digits interpreted according to a particular set of rules.

Positional notation allows numbers to be written that express all quantities, no matter how large or how small. The real value of a digit depends on its position in the number. The digits of the number 5555 are identical, yet each has a different value. To write 5555 is a compact way of writing five thousand + five hundred + fifty + five or, expressed in powers of 10, $5 \times 10^3 + 5 \times 10^2 + 5 \times 10^1 + 5 \times 10^0$. The number 10 is the base, or radix, of the decimal system. After learning a few simple rules (and memorizing or referring to some unfamiliar addition and multiplication tables), it is easy to perform calculations in any non-decimal system. This chapter is concerned with the binary number system, whose radix is 2, and the hexadecimal system, whose radix is 16.

2-3. Binary Numbers

Binary numbers are written using radix 2. That is, each column represents a power of 2, just as in decimal, each column represents a power of 10. The binary number 101101 can be written 101101_2. Its value is expressed in the equation:

$$\begin{aligned} 101101_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2 \\ &= 1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\ &= 45_{10} \end{aligned}$$

The following table lists eleven binary numbers and their decimal equivalents.

Binary	Decimal
$2^4\ 2^3\ 2^2\ 2^1\ 2^0$	
0 0 0 0 0	0
0 0 0 0 1	1
0 0 0 1 0	2
0 0 0 1 1	3
0 0 1 0 0	4
0 0 1 0 1	5
0 0 1 1 0	6
0 0 1 1 1	7
0 1 0 0 0	8
0 1 0 0 1	9
0 1 0 1 0	10

Computer people have become accustomed to referring to digits in the binary system as bits, which is a contraction of binary digits.

2-4. Converting Decimal Numbers to Binary Numbers

A simple method, suitable for converting large numbers, consists of repeatedly dividing the decimal number by 2. The remainder at any step of the division can only be 0 or 1. These remainders are the bits of binary equivalent. To illustrate, convert 37_{10} to its binary equivalent.

$$
\begin{array}{rl}
 & 37 \\
\div\ 2 = & 18 \text{ remainder } 1 = 2^0 \text{ (least significant digit)} \\
\div\ 2 = & 9 \text{ remainder } 0 = 2^1 \\
\div\ 2 = & 4 \text{ remainder } 1 = 2^2 \\
\div\ 2 = & 2 \text{ remainder } 0 = 2^3 \\
\div\ 2 = & 1 \text{ remainder } 0 = 2^4 \\
\div\ 2 = & 0 \text{ remainder } 1 = 2^5
\end{array}
$$

Binary equivalent =

2-5. Converting Binary Numbers to Decimal Numbers

The obvious method for binary to decimal conversion is to select the one bits in the binary number and convert each one to decimal and then add the results together.

$$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

$$
\begin{aligned}
37_{10} &= 1\ 0\ 0\ 1\ 0\ 1 \\
&= 1 \times 2^5 + 1 \times 2^2 + 1 \times 2^0 \\
&= 32 + 4 + 1 \\
&= 37_{10}
\end{aligned}
$$

2-6. Binary Arithmetic

Binary arithmetic operations are much simpler to perform than decimal number system operations. So much simpler that the advantage of using fewer digits to express a given value in the higher radix is more than offset. The rules of arithmetic are identical in both systems.

2-7. Binary Addition

All of the possible combinations that can occur when two bits are added are shown in the following addition table:

0	0	1	1	
+0	+1	+0	+1	
0	1	1	0	with a carry of 1

A carry 1 bit is produced from the addition of 1 and 1. Binary carries are treated in the same way as decimal carries; they are carried over to the left. In decimal, $1 + 1 = 2_{10}$, but since 1 is the largest bit, 2 must be written as 10_2. Example:

Decimal	Binary
15	1111
+ 7	+ 111
22	10110

2-8. Binary Subtraction

As in the binary addition table the binary subtraction table contains only four entries:

0	1	1	0	
−0	−0	−1	−1	
0	0	0	1	with a borrow of 1

A borrow must be made in order to subtract a larger bit from a smaller bit, just as in a decimal subtraction. Since there are only two bits, this only happens when 1 is subtracted from 0. In this case a 1 is borrowed from the next column to the left. All binary subtraction is performed according to the subtraction table. Example:

Decimal		Binary
15	minuend	1111
− 7	subtrahend	− 111
8		1000

Decimal	Binary
15	1111
− 6	− 110
9	1001

The arithmetic used in most computers performs subtraction in a different way than we are accustomed to using for decimal arithmetic. The method used is called the complement method. Its advantage lies in simpler physical circuitry to obtain the same result.

Here is how the complement method would work in the familiar decimal system. First, form the ten's complement (in binary we would form the two's complement). To form the ten's complement, subtract each digit from 9, forming the nine's complement, and then add one to the number as a whole: thus the ten's complement of 0123456789_{10} is

```
 9999999999
-0123456789
------------
 9876543210   nine's complement
       +1
------------
 9876543211   ten's complement
```

Then, subtracting a subtrahend from a minuend is simply adding the minuend complement.

Example: Subtract 56_{10} from 231_{10}

NORMAL		COMPLEMENT	
231	minuend	231	
−056	subtrahend	+944	(ten's complement)
175		(1)175	

Notice that the carry digit is ignored in the complement method. The subtrahend is the smaller of the two numbers. If not, invert the problem and change the sign of the result.

So the rule for ten's complement subtraction is

Add the ten's complement of the subtrahend to the minuend, ignoring the carry digit.

You can see that in the decimal system the ten's complement system is cumbersome.

The binary number system used by computers, however, makes subtraction by complementing simple. first, form the two's complement. Subtract each digit from 1, forming the one's complement, and then add one to the number as a whole:

```
 11111111
-00000101
----------
 11111010   (1's complement)
       +1
----------
 11111011   (2's complement)
```

Then, subtracting a subtrahend from a minuend is simply adding the minuend complement:

```
 00001010
+11111011   (2's complement)
----------
 00000101
```

As before, the carry bit is ignored.

In fact, subtraction in the MCS-48 family of computers is explicitly programmed by the complement method. Suppose you wanted to subtract A from R0, leaving the answer in A. You would program

```
CPL A      ;forms 1's complement of A.
INC A      ;now 2's complement of A.
ADD A,R0   ;A now contains the desired subtracted result.
```

There need not be a subtract (SUB) instruction.

2-9. Binary Multiplication

There are two simple, easy-to-remember rules for binary multiplication:

1. The product of $1 \times 1 = 1$.
2. All other products = 0.

0	1	0	1
×0	×0	×1	×1
0	0	0	1

The reason for the simplicity of binary multiplication is readily apparent. Any number, digit or bit multiplied by 0 produces a product of 0. The simple procedure of binary multiplication is illustrated in the following example:

Decimal	Binary	
7	111	multiplicand
× 5	×101	multiplier
35	111	
	000	partial products
	111	
	100011	product

Binary multiplication involves a series of shifts and additions of the partial products. The partial products are easily found since they are equal to the multiplicand or to 0. Every 1 bit in the multiplier gives a partial product equal to the multiplicand shifted left the corresponding number of places. Every 0 in the multiplier produces a partial product of 0. Each partial product is shifted left one position from the preceding partial product, the same as in decimal arithmetic.

It is useful to remember that shift operations are used to multiply or divide binary numbers by powers of 2 (not multiples of 2). A left shift of one position multiplies by 2; a left shift of two bit positions multiplies by 4. Similarly, a right shift of one position divides by two (i.e., multiplies by 1/2); a right shift of two positions divides by four.

2-10. Binary Division

Binary division is performed in much the same way as decimal long division. The process is much simpler, since there are only two rules in binary division.

$\frac{0}{1} = 0$	$\frac{1}{1} = 1$

Division by 0 ($1 \div 0$, $0 \div 0$) is meaningless in any numbering system. The following examples illustrate the binary division process:

Decimal	Binary
3	11
3) 9	11) 1001
9	11
	011
	11
7	111
4) 28	100) 11100
28	100
	110
	100
	100
	100

Decimal	Binary
12	1100
11) 132	1011) 10000100
11	1011
22	1011
22	1011

So, a computer does division in the reverse way as multiplication, by a series of subtractions and right shifts to provide partial dividends as opposed to a series of additions and left shifts to provide partial products.

2-11. Hexadecimal Numbers

The principal drawback of binary notation is the relative length of the numbers. It is tedious to write, and so more vulnerable to error.

One shorthand method of expressing any group of four bits is the hexadecimal number system. This is not a code, merely a means of replacing four consecutive bits by a single character. Since any four bits may represent the numbers 0 through 15, then 16 single-digit numbers are required to replace the 16 binary numbers. For convenience, hexadecimal numbers are symbolically represented by a set of familiar characters, arranged in a familiar order.

Binary	Decimal	Hexadecimal
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

Since data is often represented by binary numbers in some codes, hexadecimal notation can be used to express data. Prompt 48 uses 8-bit bytes, which can be expressed in two hexadecimal characters. The computer still reads only binary numbers; hexadecimal is the user's shorthand, not the computer's. The smallest hexadecimal number is 00_{16} (00000000_2) and the largest is FF_{16} (11111111_2).

When making translations, you may find it helpful to divide each 8-bit byte into two 4-bit nibbles. The left nibble represents the left (most significant) hexadecimal digit, and the right nibble represents the right (least significant) hex digit. For example, 01110011_2 (115_{10}) might for convenience be written:

0111	0011
7	3

i.e., 73_{16}, which looks a lot like 73_{10} but is larger in value.

For another example, 11011011_2, which translates into decimal as 219_{10}, can be translated into "hex" like this:

1101	1011
D	B

If thinking of DB_{16} as a number somewhat larger than the number of bones in your body is hard, you can calculate it using an equation much like the one used to find the decimal value of binary numbers, thusly:

$$\begin{aligned} DB_{16} &= 13 \times 16^1 + 11 \times 16^0 \\ &= 13 \times 16 + 11 \times 1 \\ &= 208 + 11 \\ &= 219_{10} \end{aligned}$$

the decimal value stated previously. In hex, there are only two digits to contend with, and each of those could be looked up in a table and thereby translated from binary in one step. As you can see, there is no direct way to divide a binary number into decimal nibbles. That's why Prompt 48 uses a hexadecimal display and keyboard.

Since hexadecimal notation is merely a shorthand for binary notation, hexadecimal arithmetic—addition, subtraction, multiplication, and division—is simply binary arithmetic. Thus,

Binary	Decimal	Hexadecimal
1001	11	B
+1010	+10	+A
10011_2	21_{10}	15_{16}
1011	11	B
−1010	−10	−A
1	1	1

Prompt 48 has a built-in hexadecimal calculator which facilitates hex addition and subtraction.

Throughout this book, numerical values are stated in decimal numbers without subscript, and program addresses and steps are stated in hexadecimal numbers without subscript. Some books use suffix H to indicate hex, D for decimal, and B for binary.

2-12. Electrical Representation of Binary Digits

So far, the bit has been discussed in terms of 1 or 0. This is fine for arithmetic and logic representation using a pencil and paper, but a computer is an electronic device, and needs two signal states that:

a. Can be represented by high speed circuits.

b. Can be readily distinguished.

c. Cannot be confused.

In general, computers use voltage levels to represent binary digits. The level may be present for a relatively short time period (or pulse) or a longer time period (which still may be a pulse or a level).

2-13. Positive True Logic

One representation of a logic level is termed positive true, and the companion voltage levels are +5 Vdc and 0Vdc, such that:

+5V = 1 = HIGH = TRUE
0V = 0 = LOW = FALSE

If the output of a logic element (circuit) is +5V, that output may be referred to as logic 1, or high, or true, depending upon the function of the logic element, i.e., whether it represents data in some form, or a timing or control function. Conversely, when the output is 0V it may be referred to as a logic 0, or low, or false.

2-14. The Inverse State (Negative True)

Certain logic elements have two outputs, the one being the inverse of the other in terms of voltage levels. In certain cases a level is purposely inverted because it is easier to use its inverse. What does this mean?

Consider a logic element that has two outputs, which are named for schematic or illustrative purposes. Now suppose that the logic element performs a control function and that the control function is termed Fetch. The mnemonic for one output could be FETCH, and by adopting the bar convention the other output would be FETCH/. How then is the Fetch control function expressed in these terms?

FETCH = 5V = HIGH = TRUE FETCH/ = 0V = LOW = TRUE	The Fetch control is applied

FETCH = 0V = LOW = FALSE FETCH/ = 5V = HIGH = FALSE	The Fetch control is not applied

Since the two signals are derived from the same logic element, they will always be opposite, the one being the inverse of the other. However, you cannot say that if FETCH = TRUE, then FETCH/ = FALSE. Both levels must be either true or false at the same time. The foregoing applies to any signal or bit that has dual representation.

CHAPTER 3
HOW THE INTEL MCS-48 CHIP-COMPUTERS WORK

3-1. Historical Perspective

The Intel MCS-48 Chip-Computers are truly computers-on-a-chip, unlike earlier "microprocessors." Within this single-chip microcomputer are included all the computer building blocks which have traditionally come to be regarded as basic: Central Processing Unit, Memory, and Input/Output.

The concepts leading to present-day computers date back as far as the 1830's, when Charles Babbage envisioned his "Analytical Engine." Babbage's design included all the major components of a general-purpose digital computer. He foresaw that its "store" (memory) should hold a thousand 50-digit numbers. Its "mill" (processor) would perform operations on the information and return the results to the "store." Babbage's concept was complete and accurate, for as in modern-day computers, it included "sequence mechanisms" which would select the proper numbers from the "store" and instruct the "mill" to perform the proper operation. But mechanical technology (later joined by electrical) required one hundred years to realize a working computer according to Babbage's conception.

This was the relay-powered "Complex Computer" built by Dr. George R. Stibitz at Bell Laboritories around 1939. Stibitz used a roomful of reliable, proven telephone relays to perform a limited repertoire of arithmetic operations. It worked, and was very fast alongside the manual calculation methods available to mathematicians in 1939. It was not a general-purpose machine.

3-2. The Harvard Architecture

The great technical visionary, Howard Aiken, conceived that the technique of Stibitz could be extended to fulfill Babbage's dream of a practical, general-purpose computer. His conception was of an electronic machine with vacuum-tube memory banks, used to store both numerical data and changeable programs for the processing of that data. His particular design called for split, independent memories for "data" and "programs." He wrote the specification for such an "Automatic Sequence-Controlled Calculator" in 1937.

Seven years later, the development and manufacturing skills of IBM Corporation successfully completed and installed this system, Mark I, on which the Harvard Computation Laboratory was founded. It was 51 feet (15.5 Meters) in length. Its information was input by four paper-tape readers. Three were dedicated to data, one to programs, whose instructions were coded in the sequence "source, destination, operation." The Mark I was very slow by modern standards: about 1/3 second was required to execute a single ADD instruction.

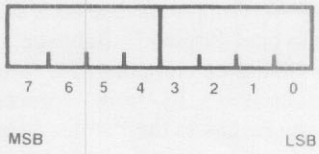
3-3. Princeton Heard From

A computer named EDVAC was the result when mathematician John von Neumann and his colleagues at Princeton constructed a machine for the U.S. Army. EDVAC could store 4K of a mixture of 40-bit data words and program instructions in its vacuum-tube memory matrix. The principal von Neumann introduced is that of numerical coding of programs, in exactly the same format as data, stored side-by-side in the same memory. This was a technique of such power and flexibility (especially so in an era when memory was costly) that it has been adopted and used virtually universally. Intel's 8008 and 8080 series of microprocessors are designed fundamentally around the Princeton architecture; they are "von Neumann" machines, employing a "monomemory" addressing scheme. On the other hand, the 4004 and 4040 are "Aiken" machines, featuring the Harvard Architecture, employing separate program and data memories.

3-4. The MCS-48 Architecture

3-5. Bits, Bytes, and Where You Can Put Them

The basic unit of information in virtually any computer system today is the bit. A bit is a binary (base 2) digit; that is, it can be either a 0 or a 1, represented in a computer as a low or a high voltage level. In the MCS-48 series computer systems, bits are handled in groups of eight. Space for data is allocated in these eight-bit bytes. For easy identification, the bits in a byte are numbered according to their position, or power of 2, from 0 through 7, or least significant bit (LSB) to most significant bit (MSB), thus:



An eight-bit byte is conventionally broken up into four-bit half bytes, called nibbles. A nibble, containing four bits, can represent $2^4 = 16$ different numbers, from 0000 to 1111. For programming convenience, four-bit nibbles are usually represented as a single hexadecimal digit (base 16), from 0 to F_{16}. To understand the inner workings of the computer you need to think binary, but when you are writing programs for the MCS-48 chip-computers you'll be writing hex numbers, rather than bits.

A register is a place to store binary data so it can be worked with. Most MCS-48 registers are 8-bits wide (one byte). Each MCS-48 Chip-Computer contains Register Memory, Data Memory, and separate Program Memory, thereby reintroducing the Harvard Architecture. The MCS-48 also retains the Princeton concept of program instructions coded in the same numerical format as data. Program memory is thus also organized as 8-bits (one byte) wide per location.

3-6. Accumulator

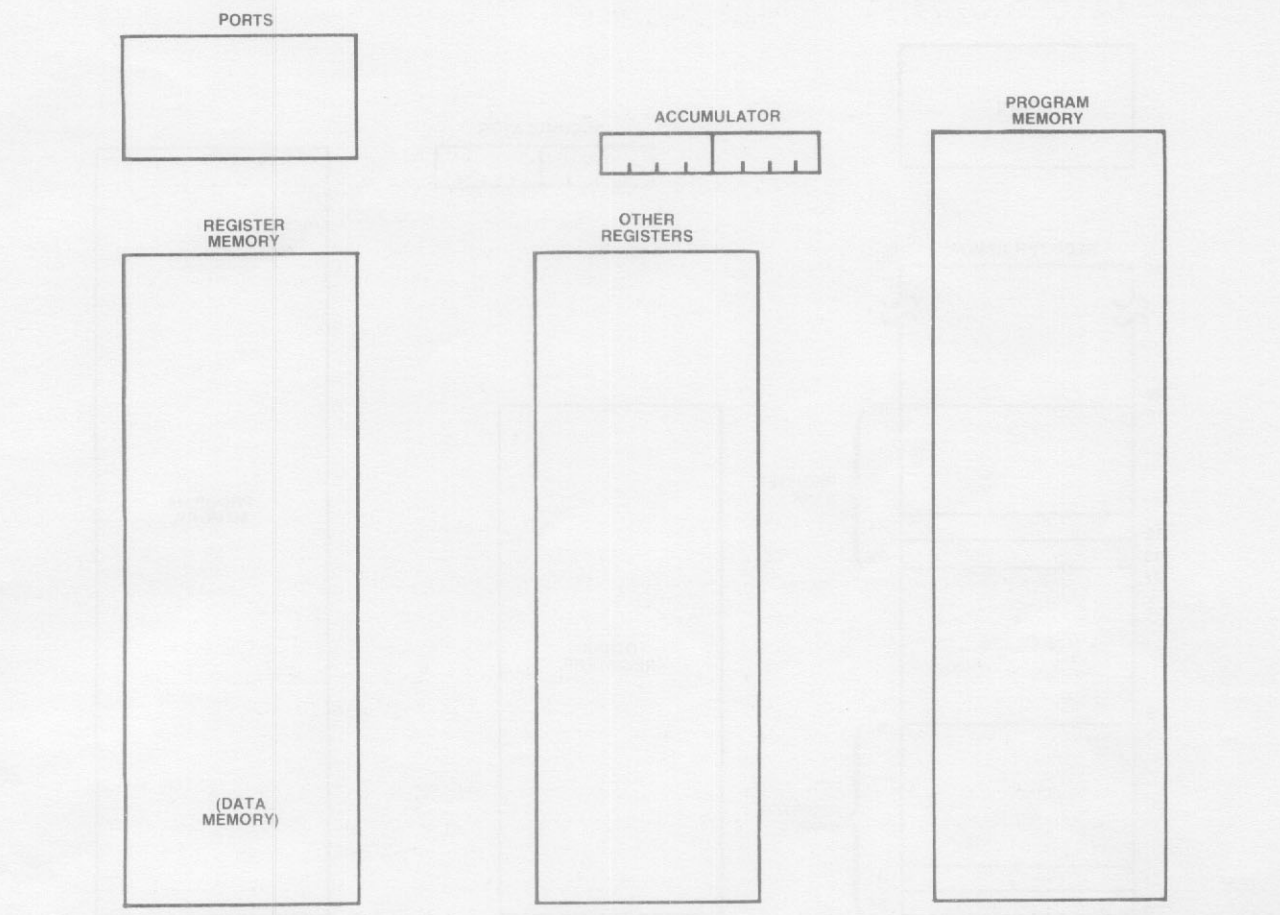
The first register to be explained is the accumulator, designated A. An accumulator is something like the display register in an electronic calculator. The accumulator is the focal point of a majority of the instructions the computer can execute. Most arithmetic and logical functions are performed on the data within the accumulator, or between the accumulator data and the contents of other data sources (registers and data memory). The accumulator is also the channel through which all data is transferred to and from external devices, and can be used to access data contained in program memory.

We will illustrate the architectural features of the MCS-48 family with a device known as the Micromap, which will gradually increase in complexity until it becomes a quick reference to the features and capabilities of the MCS-48. The first Micromap, emphasizing the accumulator, appears below.

3-7. Register Memory, Working Registers, and RAM Pointers

The MCS-48 Chip-Computers contain 64 8-bit bytes of register memory, numbered $00\text{-}3F_{16}$. These registers are divided into two major types, working registers and data storage registers. The working registers have the special capability of being directly accessible through a wide variety of register-accumulator instructions and register-only instructions.

The working registers are divided into two banks of eight registers each, designated R0, R1, . . . , R7, of which only one bank is directly accessible at any given time. Working Register Bank 0 (RB0) is found in locations 0-7 of the register memory, and working Register Bank 1



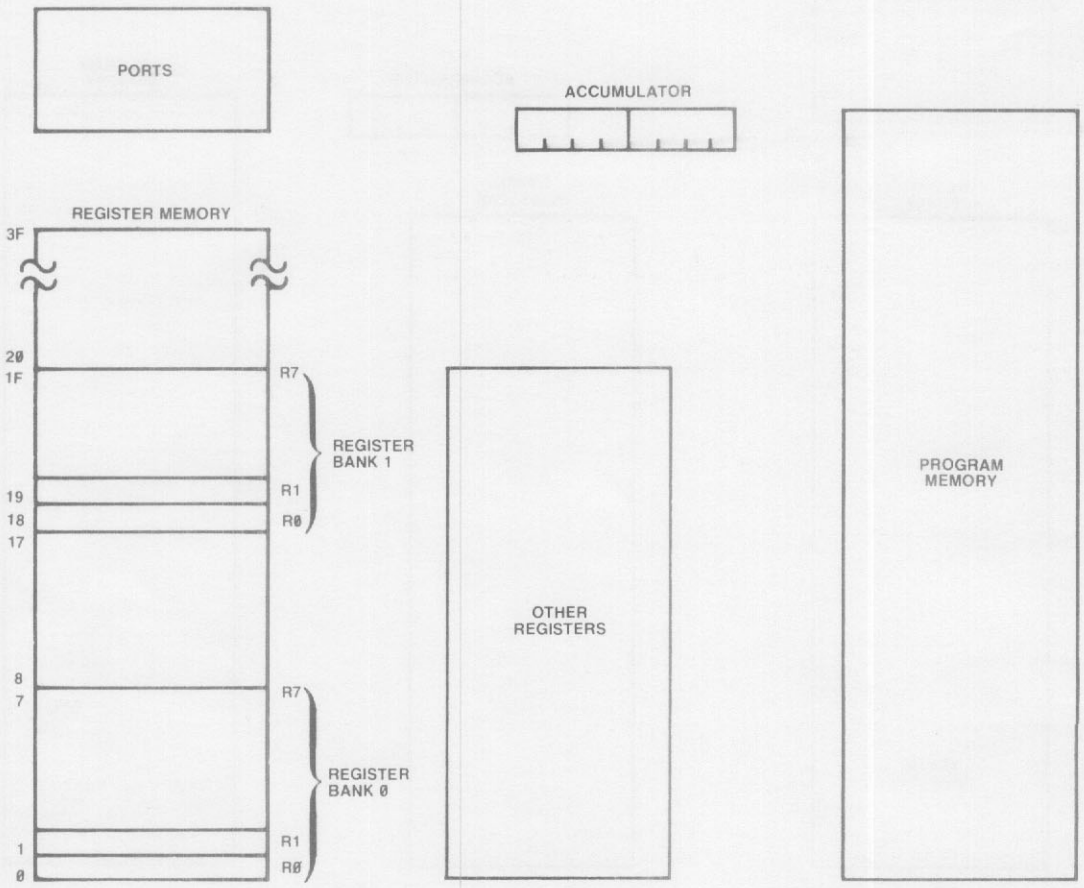
(RB1) in locations $18_{16} - 1F_{16}$. The bank currently being used is selectable under software control (see paragraph 3-17).

Two working registers in each bank, R0 and R1, are also called RAM Pointers. Data storage registers are only accessible through the use of the RAM Pointers. The RAM Pointers can (in addition to the general capabilities of work registers) also function as "index" registers. That is, they can contain the address (register number) of a byte of the register memory whose data is to be accessed through certain instructions.

3-8. Program Memory and Program Counter

Program memory, like register memory, is a place to put information; in this case, the instructions to be carried out by the computer. In MCS-48 computers program memory is 8 bits (one byte) wide. In the 8048 and 8748, there are 1024 (1k) bytes of program memory on-chip, addressed as locations $000\text{-}3FF_{16}$.

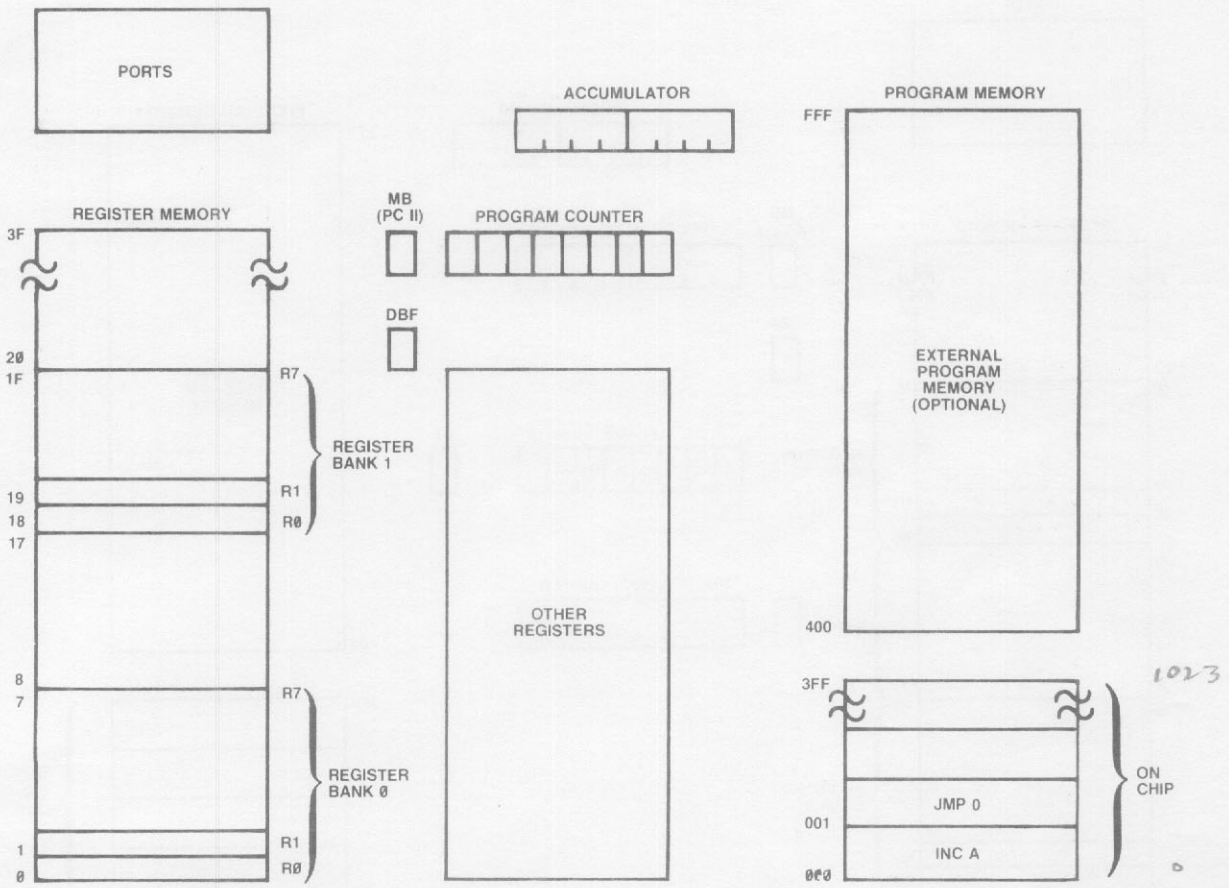
The program memory is accessed by means of the program counter. The program counter is a 12-bit register containing the address of the next instruction to be executed by the computer. Most instructions are executed sequentially in ascending addresses of program memory. That is, the program counter is "incremented" after each instruction. Breaks in the normal sequence of program execution are achieved through "jump" commands, which load the program counter with an address other than that of the next instruction in program memory. Note that a 12-bit register can address $2^{12} = 4096$ locations. The 3072 addresses not on-chip are located in external program memory, discussed in Paragraph 3-12.



The MCS-48 Chip-Computers manage program memory in 256-byte pages. The most significant hex digit of the program memory address is the page number; the entire 4096-byte address range of the MCS-48 would amount to sixteen pages. The two least significant hex digits point to 256 adjacent memory locations, numbered $X00_{16}$ – XFF_{16}, where X is the page number in hex. Memory paging is implied by the fact that only the 8 least significant bits increment automatically after each instruction. The two exceptions to this rule (the only means to cross "page boundaries") are the CALL and JMP instructions, which provide an additional 3 more significant bits of address information (a total of 11 bits). A 12^{th} and most significant bit exists in the program counter, called the Memory Bank select, or MB bit. This bit may be manipulated by software to select either of two 2k regions (upper or lower) of program memory through the Designated Bank Flag (DBF), which is moved into MB on the execution of a CALL or JMP instruction (see Paragraph 3-17).

3-9. Flags and Stack

The flags in the MCS-48 are independent on-bit registers which are used as aids to various processing tasks. Four of the flags are organized into half of the flags register which contains the processor status word, or PSW. These four are the Carry (C), Auxiliary Carry (AC), user Flag 0 (F0), and working register Bank Select (BS) flags. The C flag represents the carry (or borrow) from the last addition (or subtraction). The AC flag represents the carry from bit 3 to bit 4 of the last addition, which is needed for decimal arithmetic. F0 is set, reset, and sensed by software, and is useful as a means of communicating between two parts of a program. BS determines which working register bank is currently in use: RB0 (register

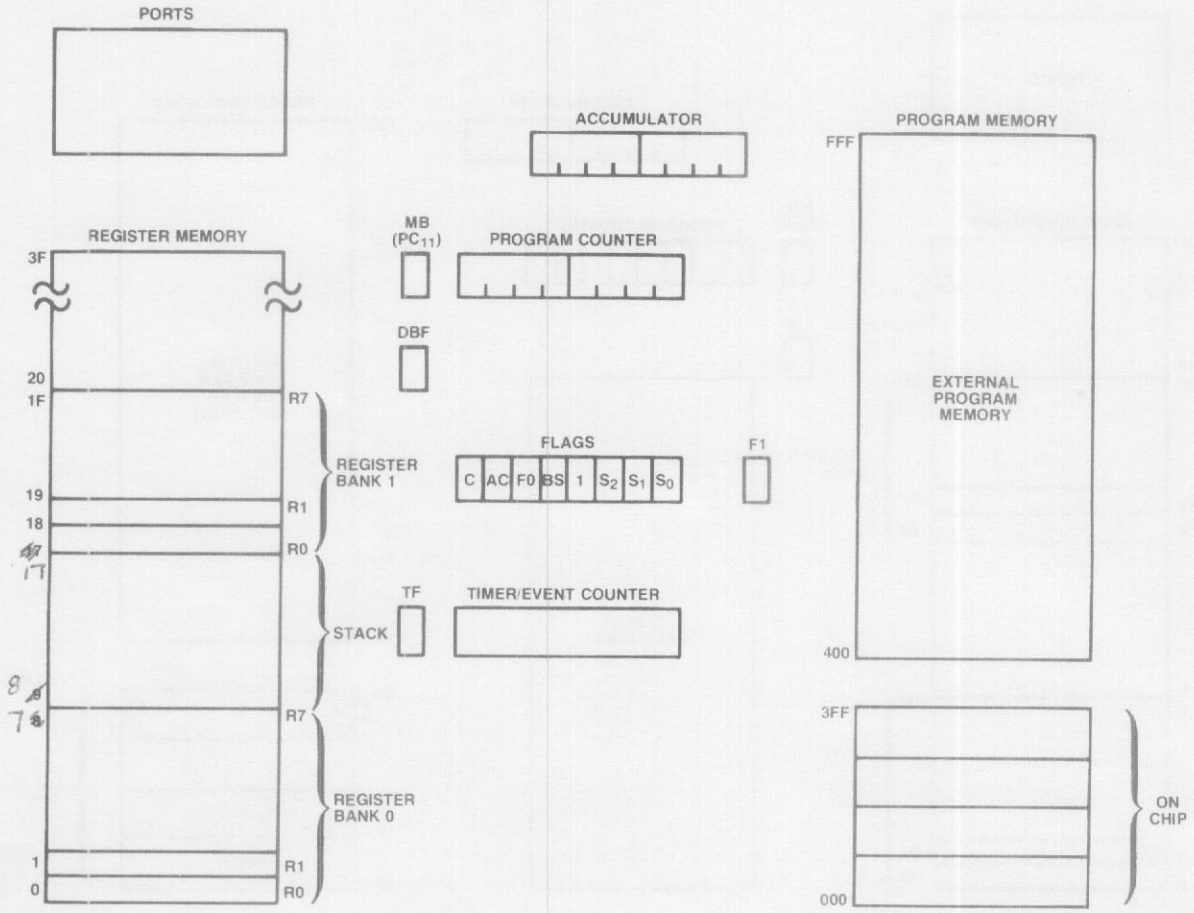


memory locations 0-7) or RB1 (locations $18_{16} - 1F_{16}$). Contained elsewhere in the MCS-48 are user Flag 1 (F1 — used like F0), the Timer Flag (TF — see Paragraph 3-10), and the Designated Bank Flag (DBF — see Paragraph 3-8).

Also stored in the flags register are the three STP bits, the stack pointer. The stack pointer is used to manage the MCS-48 stack. A stack is a splendid way to organize activities that cannot be done at the same time. Here is an example from day-to-day life. Suppose that you are writing at your desk and the phone rings. You set aside the writing (intending to return to it) to take care of the phone call. Then a second person calls. You place the first caller on hold and answer the second caller's question. Then you return to the first caller and ultimately to your writing.

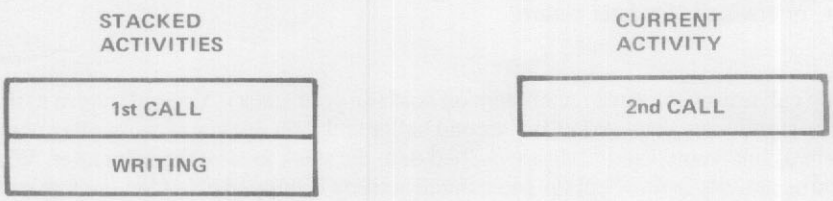
How do you organize your responses to these multiple demands? When the first phone rings you remember (perhaps on a mental list of things to do, or mental "stack") that you will return to the writing. And when the second call comes you decide that the first call can be put on hold, or stacked, for later return.

Your first call is now the most recent item on hold (on your stack). You will return to the first caller when you have disposed of the second caller and then resume writing after both calls are finished. Interrupted activities are pushed onto the stack to save them for later. When an interrupting activity is finished, the interrupted activity is popped off of the stack to restore it



for completion. The MCS-48 computers have facilities which allow a program to be interrupted, made to perform more urgent tasks, and later be returned to the original activity through the use of a stack.

In the MCS-48 Computers, the stack is implemented by saving the contents of the program counter (return address in the interrupted activity) and the C, AC, F0, and BS bits of the flags register (status of the interrupted activity). The twelve bits of the program counter and the four bits of the flags register are combined into two bytes, which are saved on the stack. The stack is a special area of register memory, locations $8_{16} - 17_{16}$. These sixteen bytes of register memory are divided into eight two-byte stack locations, or levels. This allows eight levels of "nesting," or eight interrupted activities waiting on the stack.



The stack is maintained through the use of the stack pointer (STP), the three low order bits of the flags register. These three bits can point to (address) the $2^3 = 8$ stack locations. Note that the STP bits do not form the actual address in register memory of the stack, but rather indicates the next available stack entry, called the stack "top." When STP = 000, the stack is on level 0, and the next available stack location is at register memory locations 8 and 9. Similarly, when STP = 001, the stack is on level 1, and the next location is in register memory A_{16} and B_{16}.

The format of a stack push is shown in Figure 3-1. The eight low order bits, bits 7 to 0, of the program counter, are saved in the low order byte, the lower address of the stack registers. The four flag bits are combined with the program counter bits 11-8 (including MB) to form the upper byte of the stack entry. After the transfer, 1 is added to the stack pointer to point to the next available stack entry, on the next level.

A stack pop is shown in figure 3-2. The stack pointer (STP) points to the next available stack level. First, 1 is subtracted from the stack pointer. Then the data to restore the interrupted activity is transferred from the now available stack location to the appropriate registers.

The stack is also used to manage subroutines. A subroutine is a part of a program that is used ("called") by other parts of the program. An example would be multiplication routine, which would calculate and "return" the answer, the product. As with interrupts, the status and return address are saved on the stack, and can be restored to the flags and program counter registers in order to return to the calling routine (previous activity). In most cases though, the status of the subroutine does not interfere with the main (calling) program (self-interrupted activity), so there is a special instruction to pop only the return address from the stack for use with subroutines (see Paragraph 3-17).

All this is not to say that the memory in which the stack resides is any different then the data storage registers, for they are equally accessable through the use of the RAM pointers. While the register memory is available for data storage on those levels of the stack which are not needed to monitor multiple activities, this very availability should be carefully checked. Writing a byte of unrelated data over a return address can be disastrous.

3-10. Timer/Event Counter

Each MCS-48 computer has an on-chip timer/event counter to count external signals or to generate time delays without tying up the processor. Basically, it is an 8-bit register that (when enabled) increments every time it gets an input, and sets a flag when full. The input can be either an external signal, or an internally generated signal, equal to 1/480 of the clock crystal frequency. These are the event counter and timer modes, respectively. Dividing the clock frequency by 480 means that, for example, if the system clock crystal frequency was 3 MHz, the timer would increment every .16 msec. This is equal to 32 instruction cycles. When the timer/event counter is full (all ones), the next increment resets the timer/event counter to zero, and sets the Timer Flag (TF). This flag can then be used by the software to decide whether it is time to perform a time- or external event-dependent action. The timer/event counter continues incrementing on each input, regardless of the reset when full, until stopped by software. The instructions used to control and monitor the timer/event counter are described in Paragraph 3-16 and the *MCS-48 Microcomputer User's Manual*.

40.96 ms

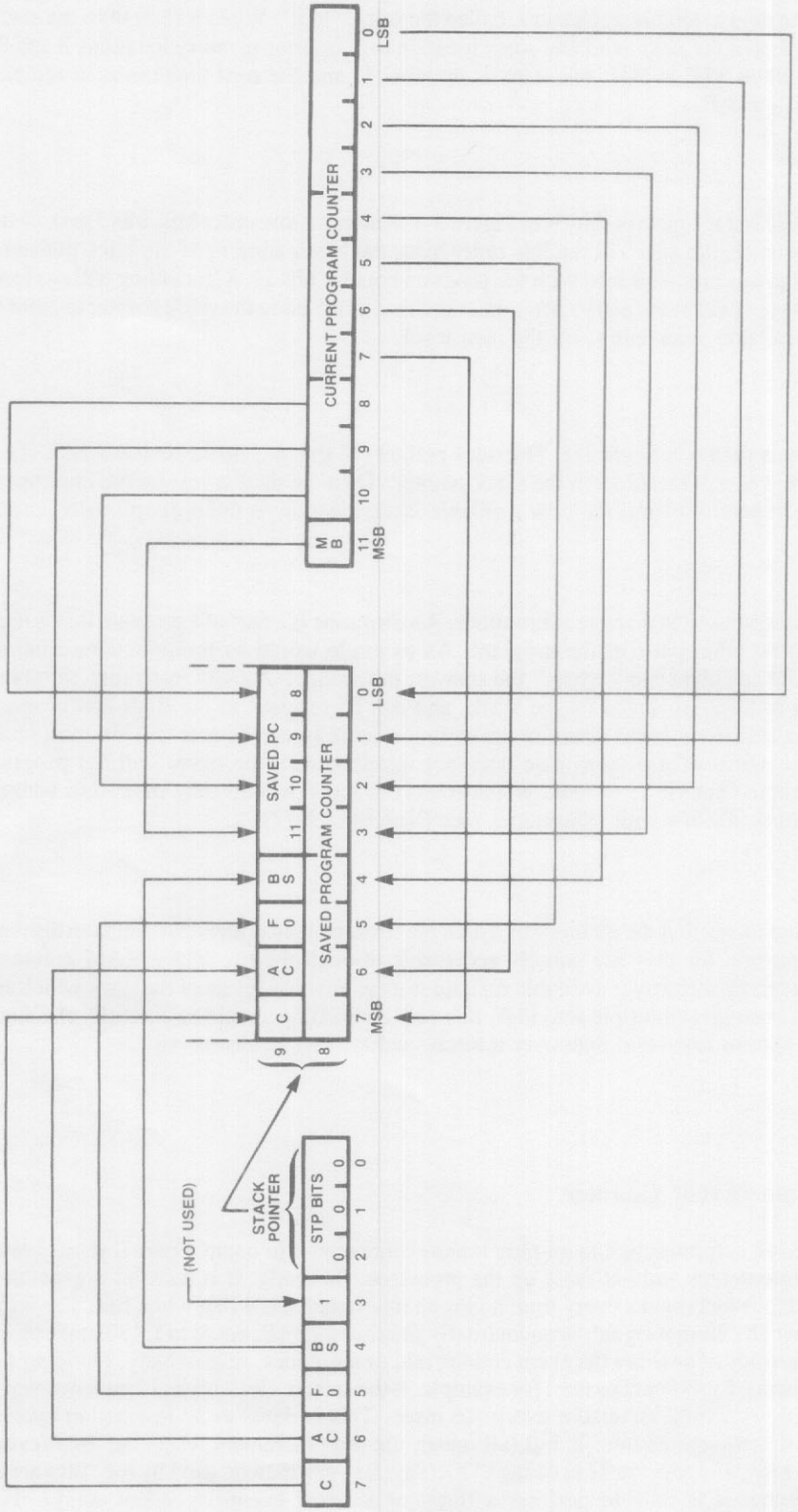


Figure 3-1. Stack Push

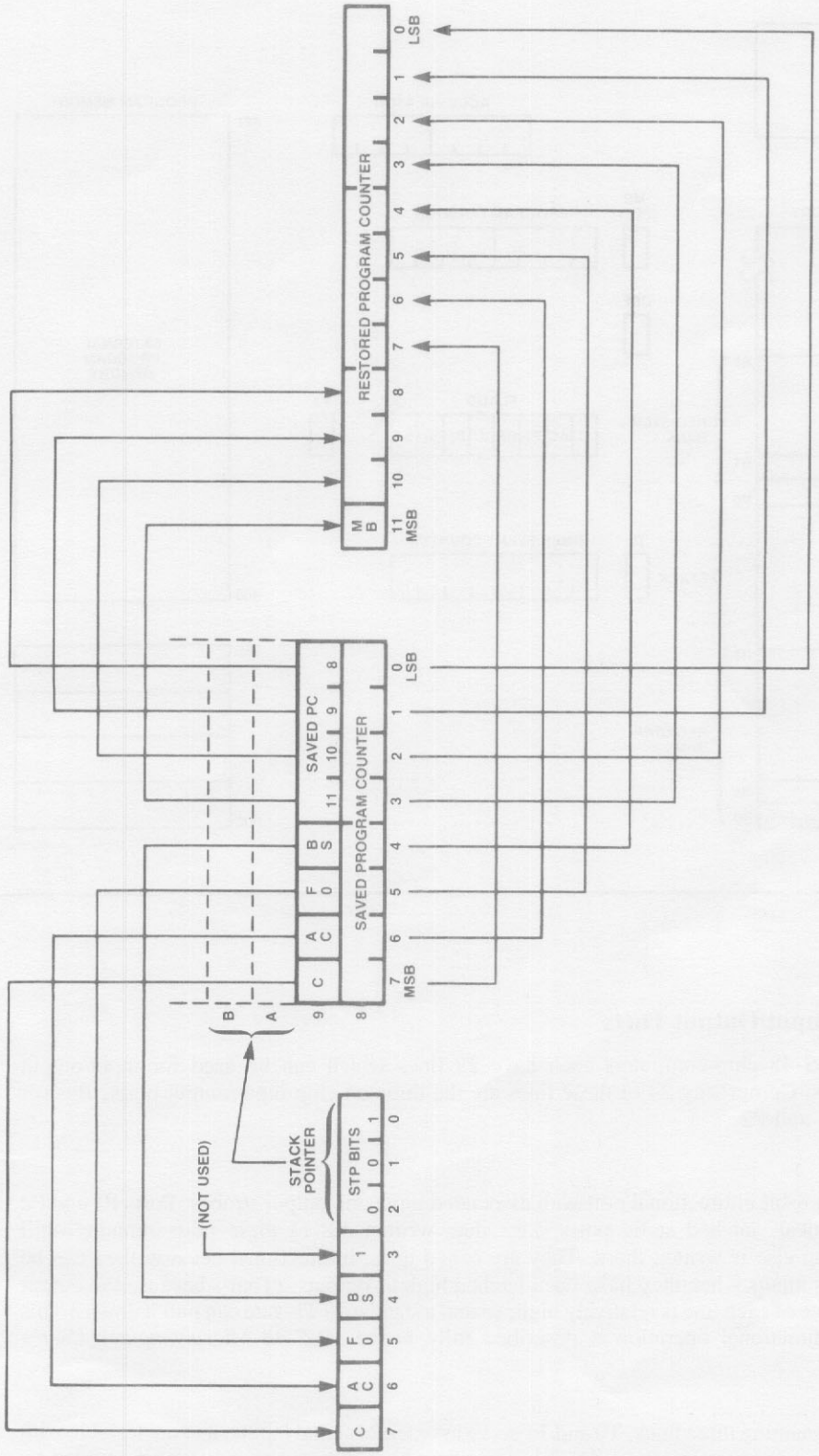
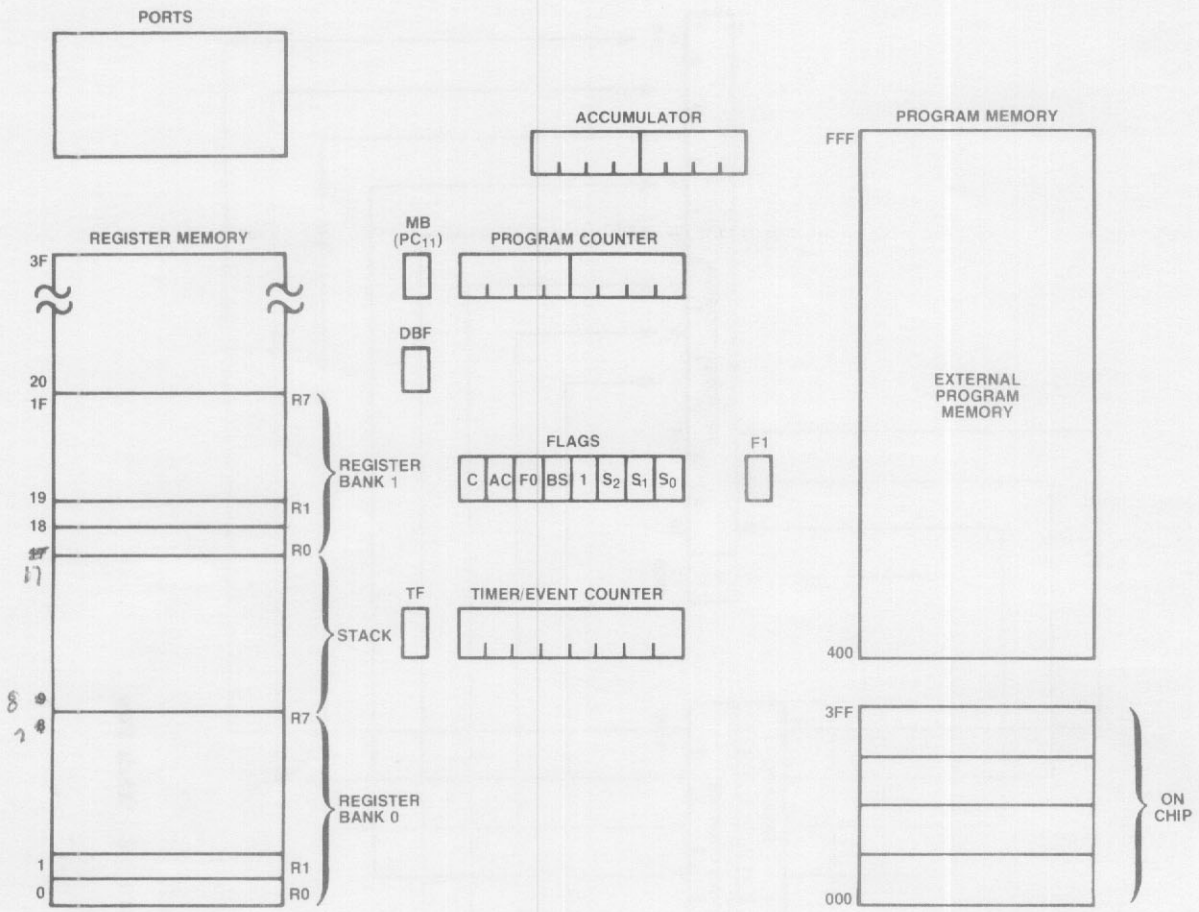


Figure 3-2. Stack Pop

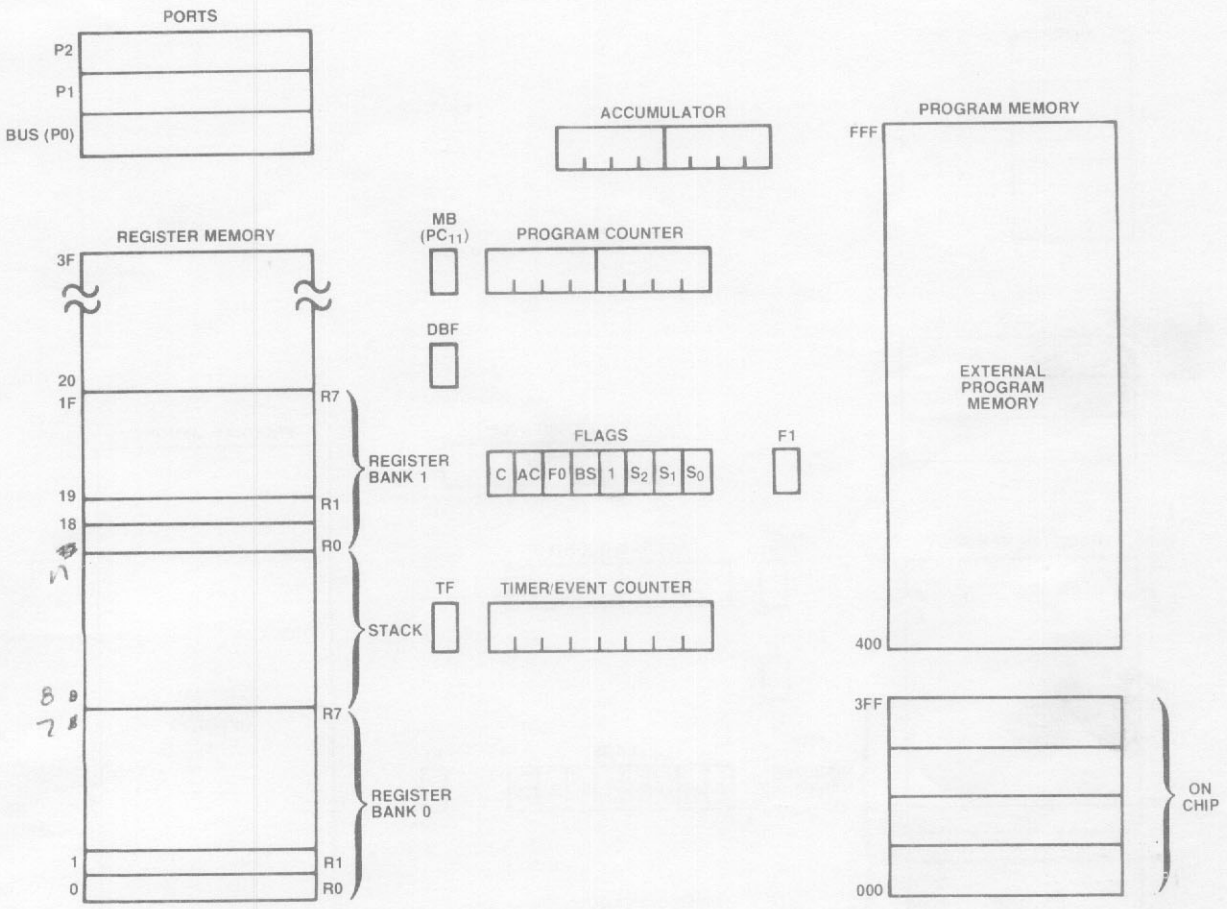


3-11. Input/Output Ports

The MCS-48 chip-computers each have 27 lines which can be used for input/output functions. Comprising 24 of these lines are the three on-chip input/output ports, Bus (or P0), P1, and P2.

Bus is an 8-bit bidirectional port with associated input and output strobes. Ports P1 and P2 are identical, latched static ports, i.e., data written out to these ports remains until something else is written there. They are called quasi-bidirectional because they can be driven as inputs when they have been latched high as outputs. (That's because the output impedance of each line is relatively high, so that a standard TTL gate can pull it down.) This quasi-bidirectional operation is described fully in the *MCS-48 Microcomputer User's Manual*.

Of the remaining three lines, T0 and T1 serve as external signal inputs, and are testable with conditional jump instructions. INT/ is an input which initiates an interrupt if enabled by software. The relevant instructions are given in the *MCS-48 Assembly Language Manual*, and the hardware operation is described in detail in the *MCS-48 Microcomputer User's Manual*.

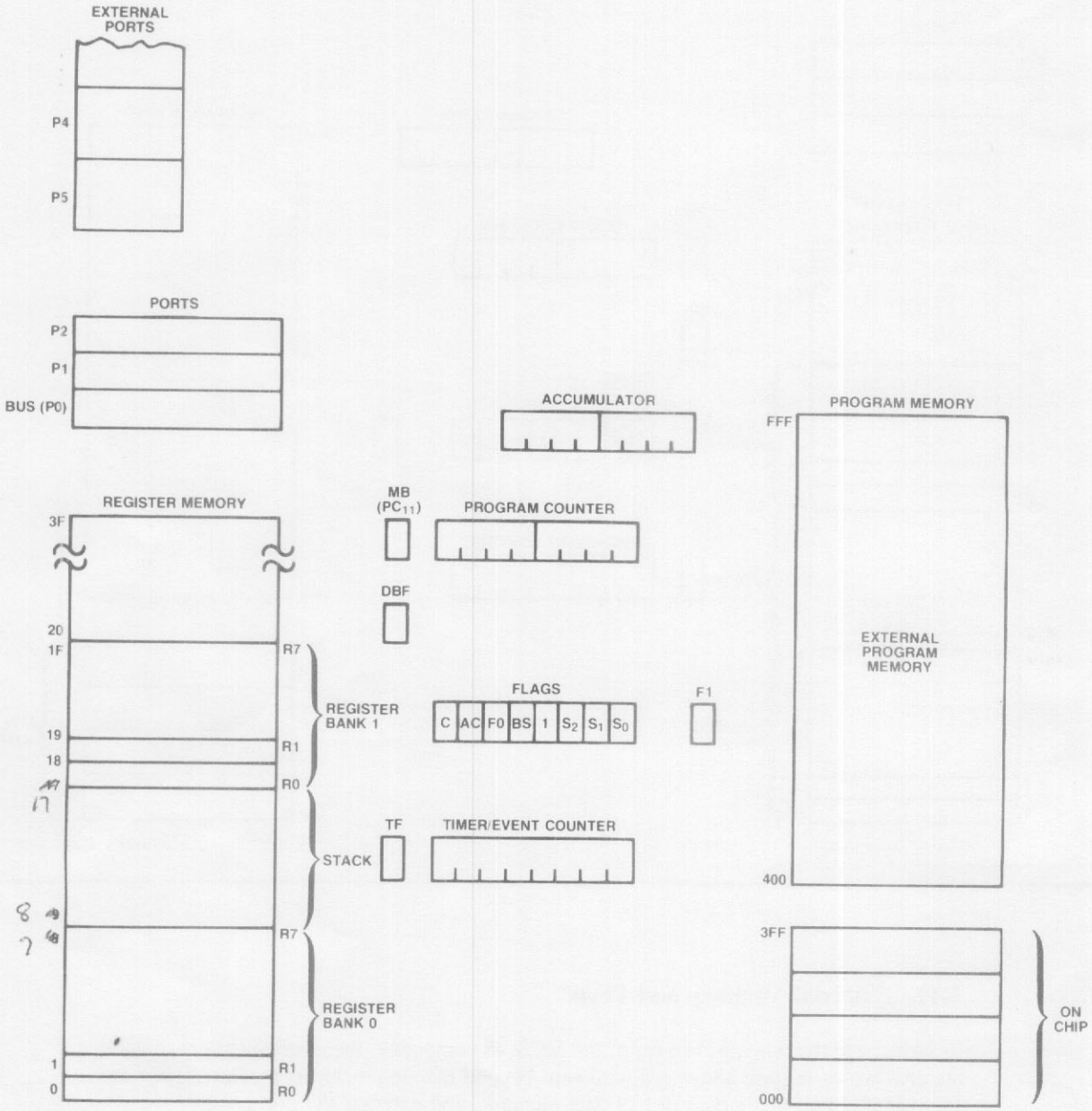


3-12. External Memory and Ports

In addition to the on-chip features of the MCS-48 computers, there are several expansion features which require additional hardware beyond the single-chip computer. These are external program memory, external data memory, and external I/O ports.

3-13. External Program Memory. If a given application requires more than the 1024 program memory bytes included on-chip, there is provision for expanding the program memory with up to 3072 additional bytes of external memory, making a total program memory of 4096 (4k) bytes possible. (For details on how to implement program memory expansion, see the *MCS-48 Microcomputer User's Manual.)*

The external program memory is treated in the same manner as in the 256 byte pages (see Paragraph 3-8). There is, however, an additional condition which must be observed when program memory exceeds 2048 bytes. This is the Memory Bank (MB) address bit, the most significant bit in the 12-bit program counter. (Details on how the MB bit is manipulated are given in Paragraph 3-17.)



3-14. External Data Memory. If the data requirements of an application exceed the capacity of the on-chip 64 bytes of register memory, up to 256 bytes of external data memory can be added. This external data memory is accessed through the accumulator, using one of the RAM pointers for addressing. (Complete hardware details for data memory expansion are given in the *MCS-48 Microcomputer User's Manual*. The instructions which read and write the external data memory are discussed in Paragraph 3-17).

3-15. External Ports. The most efficient means of I/O expansion for small MCS-48 systems is the 8243 I/O Expander Device (part of Intel's compatible MCS-48 family) which requires only 4 port lines (the lower half of Port 2) for communication with the MCS-48 Chip-Computer. The 8243 contains four 4-bit I/O ports which serve as extensions of the on-chip I/O and are referred to in software as P4-P7. The following operations may be performed on these ports:

1. Transfer Accumulator Data to Port
2. Transfer Port Data to Accumulator
3. AND Accumulator to Port (result in Port)
4. OR Accumulator to Port (result in Port)

All communication between the MCS-48 Chip-Computer and ports P4-P7 takes place through the Least Significant Nibble of Port 2 (LSN P2). LSN P2 corresponds to pins P20-P23 on the Chip-Computer. Data is transferred between the LSN of the Accumulator and the specified port (P4-P7). A 4-bit transfer from one of these ports to the LSN of the Accumulator sets the Most Significant Nibble (MSN) of the Accumulator to zero.

Hardware details as well as other options for port expansion are given in the *MCS-48 Microcomputer User's Manual*. The use of related software instructions is discussed in the *MCS-48 Assembly Language Manual*.

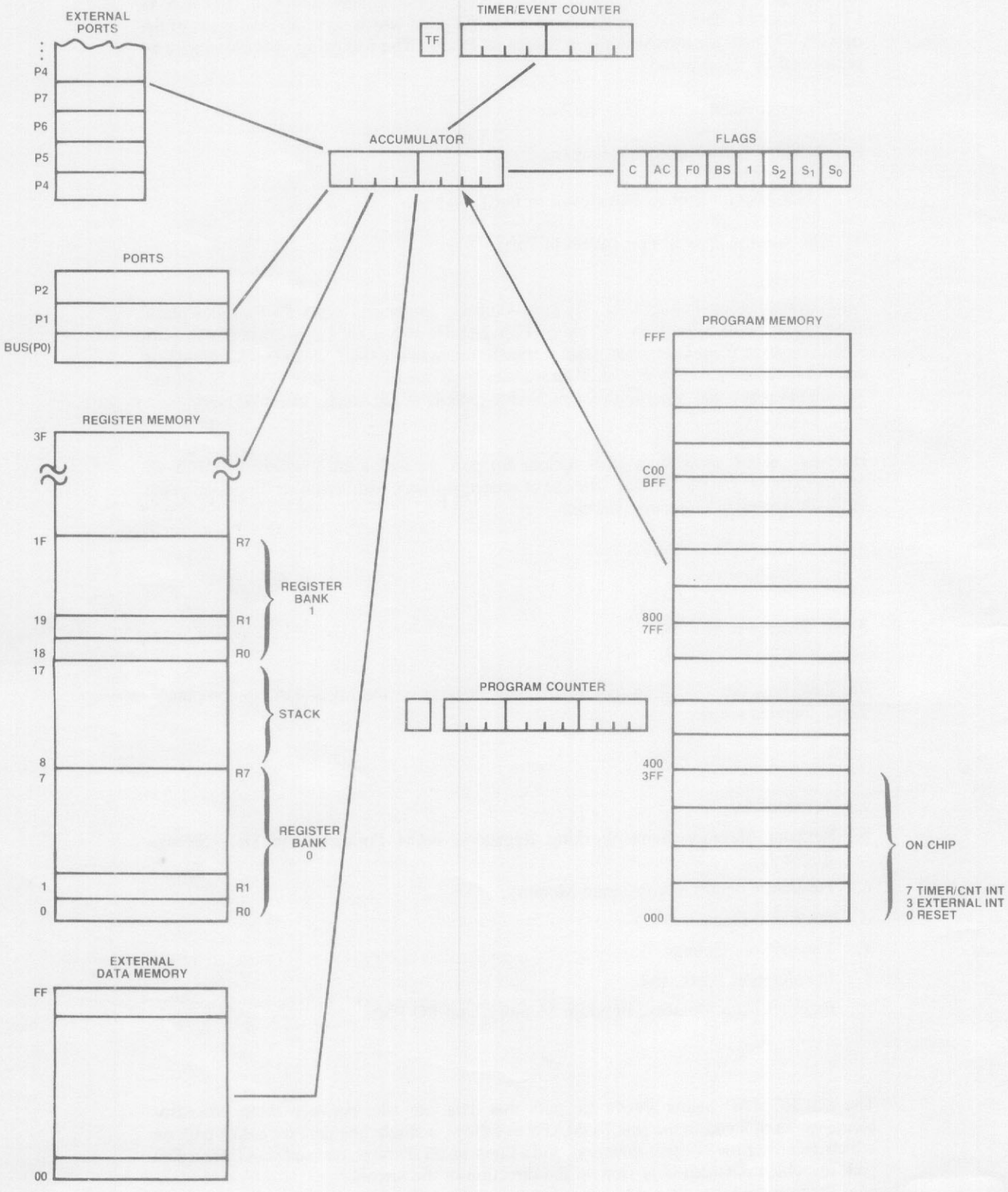
3-16. Data Paths

We have now introduced all the architectural features on the MCS-48 chip-computers. These features are the:

a. Accumulator,
b. Register Memory (with Working Registers, RAM Pointers, and Data Storage Registers),
c. Program Counter and Program Memory,
d. Stack and Flags,
e. Timer/Event Counter,
f. Input/Output Ports, and
g. External Data Memory, Program Memory, and I/O Ports.

The MICROMAP below shows the path that data can take between these processor elements. In this and in the MICROMAPS to follow, a single line denotes a data path on which data can flow in either direction, and a line with an arrow on one end stands for a data path on which data can only flow in the direction of the arrow.

Paragraph 3-17 discusses the instructions which facilitate movement along the various MCS-48 data paths, as well as all other instructions available to the MCS-48 programmer.


EXTERNAL PORTS
P4
P7
P6
P5
P4
TIMER/EVENT COUNTER
TF
ACCUMULATOR
FLAGS
C
AC
F0
BS
1
S_2
S_1
S_0
PORTS
P2
P1
BUS(P0)
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
REGISTER MEMORY
3F
1F
19
18
17
8
7
1
0
R7
R1
R0
R7
R1
R0
REGISTER BANK 1
STACK
REGISTER BANK 0
PROGRAM COUNTER
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET
EXTERNAL DATA MEMORY
FF
00

3-17. MCS-48 Instruction Set

In this section we will describe the various classes of instructions which allow data to be manipulated in the MCS-48 Chip-Computers. (For details of any specific instruction, we refer you to the *MCS-48 and UPI-41 Assembly Language Manual,* or the summaries in the *MCS-48 Microcomputer User's Manual.)*

Roughly, the MCS-48 instructions break down into four categories:

1. Accumulator Instructions
2. Register-Accumulator Instructions
3. Input/Output Instructions
4. Control Instructions

The Micromaps which illustrate these four categories use the following terminology:

R07 represents any one of the working registers, R0, R1, . . . , R7, in either Working Register Bank RB0 or RB1. (Which bank depends on the status of the BS flag bit.)

R01 can be any of the four RAM pointers, R0, R1 (Register Bank 0), and R0, R1 (RB1).

@R01 is the data memory location pointed to by the current register R01; that is, the two-hex-digit contents of R01 represents the register number in register memory, or the address in external data memory.

P12BUS represents Port 1, Port 2, or BUS (Port 0), the three ports implemented on-chip in the MCS-48 family.

P47 represents Port 4, 5, 6, or 7, the external I/O ports which can be added with very little additional hardware.

3-18. Accumulator Instructions

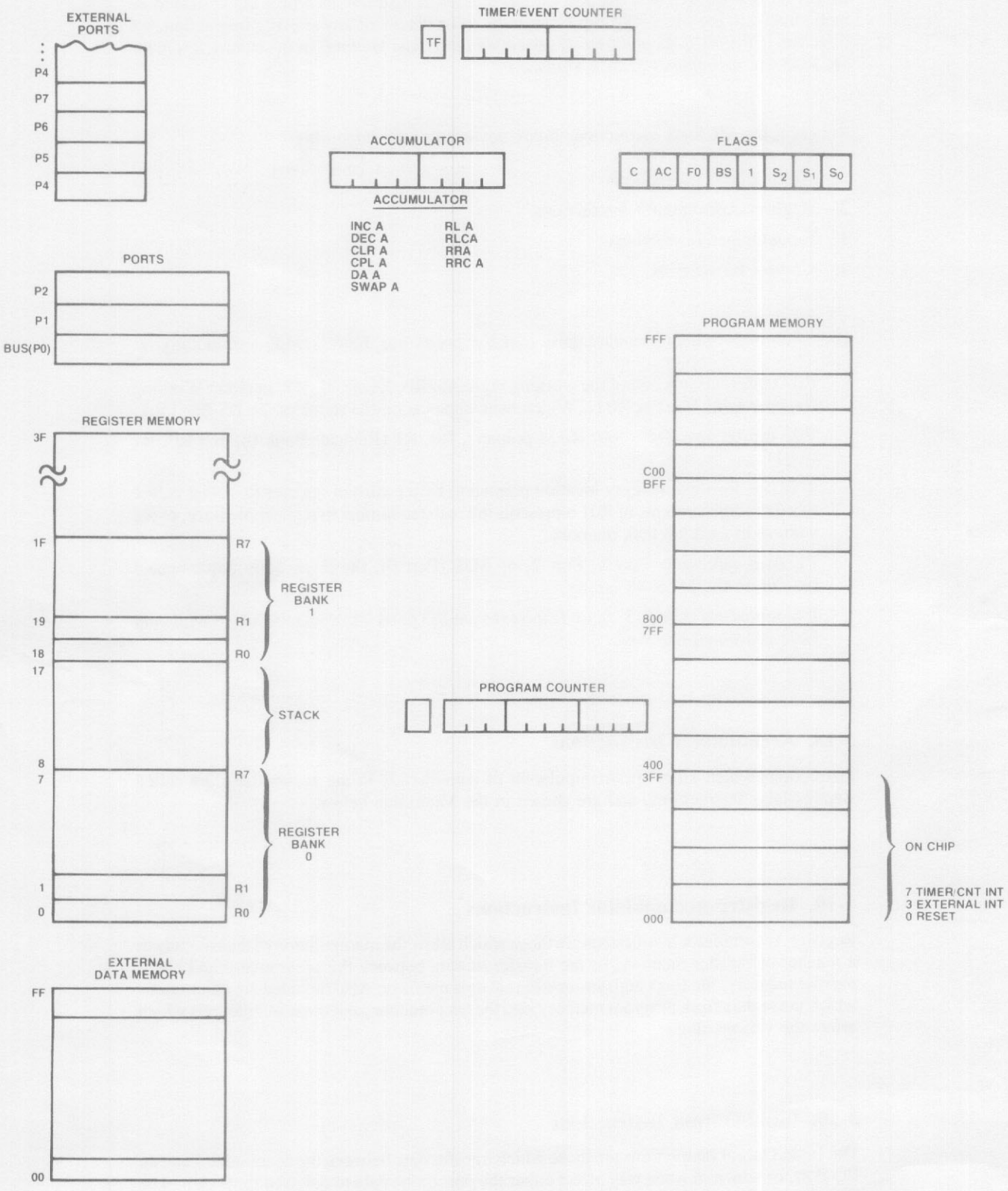
Instructions which allow the manipulation of data already in the accumulator are called Accumulator Instructions, and are shown in the Micromap below:

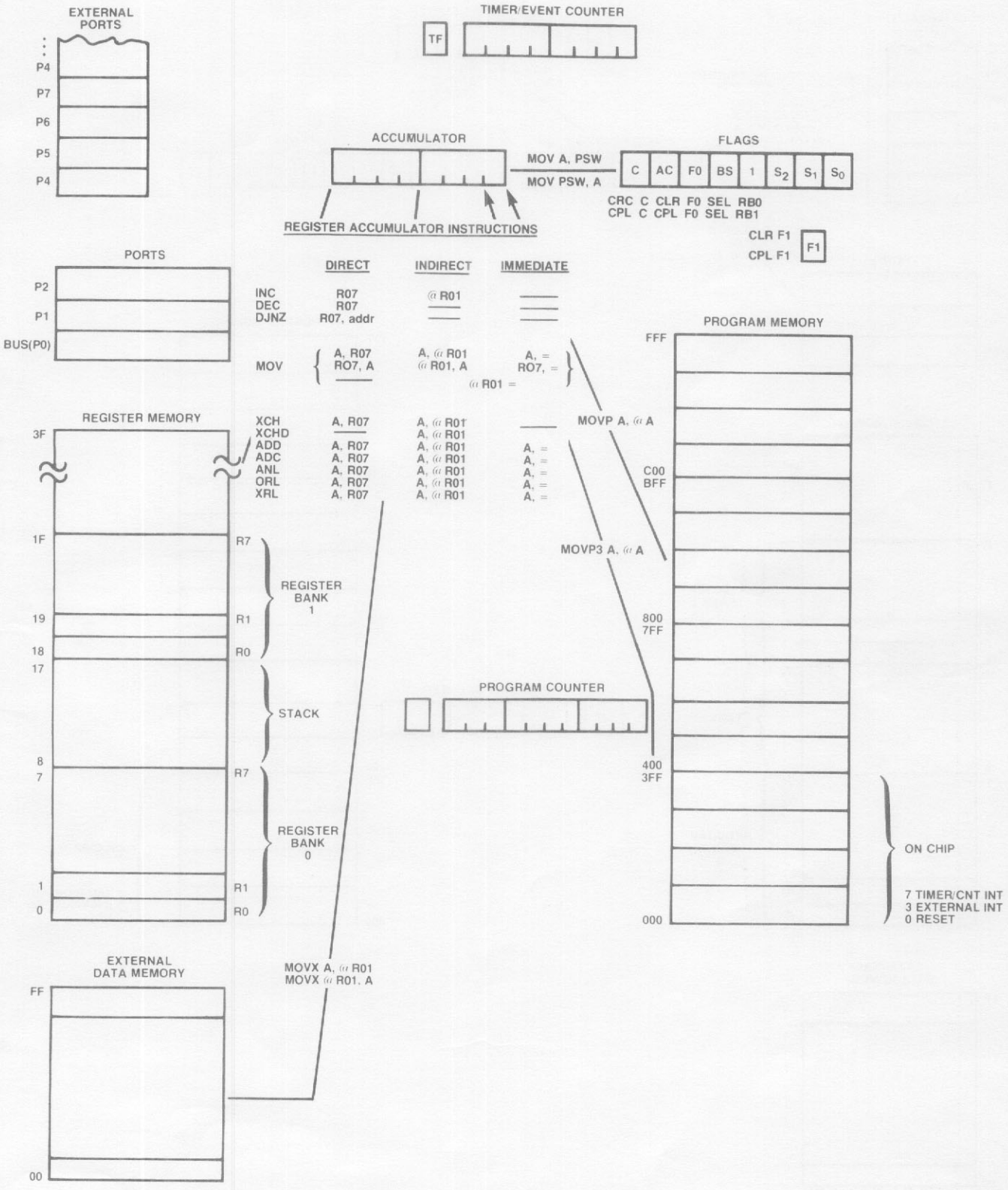
3-19. Register Accumulator Instructions

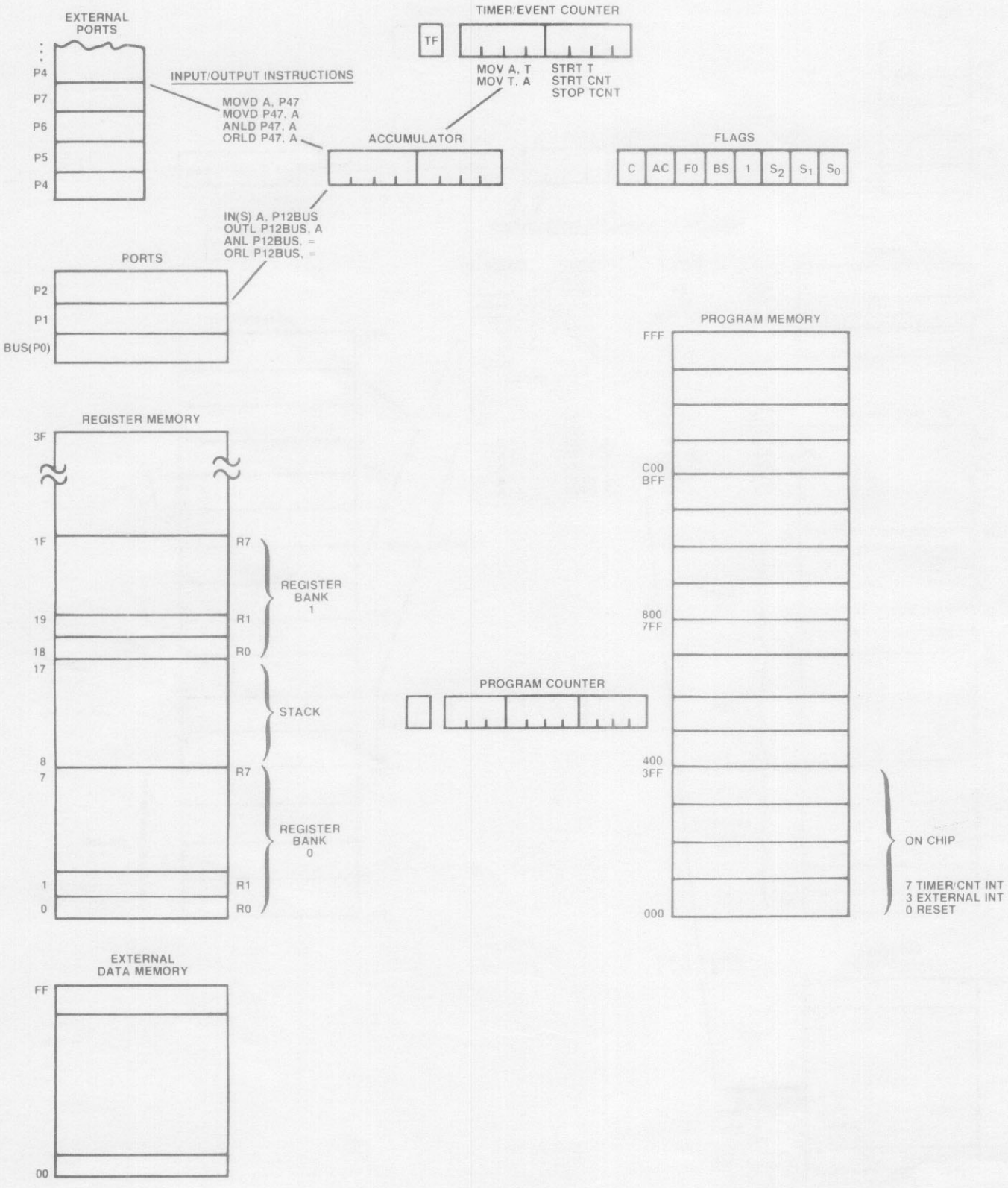
Register Accumulator Instructions are those which allow the manipulation of data already in a register of register memory, or the transfer of data between the accumulator and either register memory, the flags register, or external data memory. Also included are instructions which move data from program memory into the accumulator, and those instructions which affect the various flags.

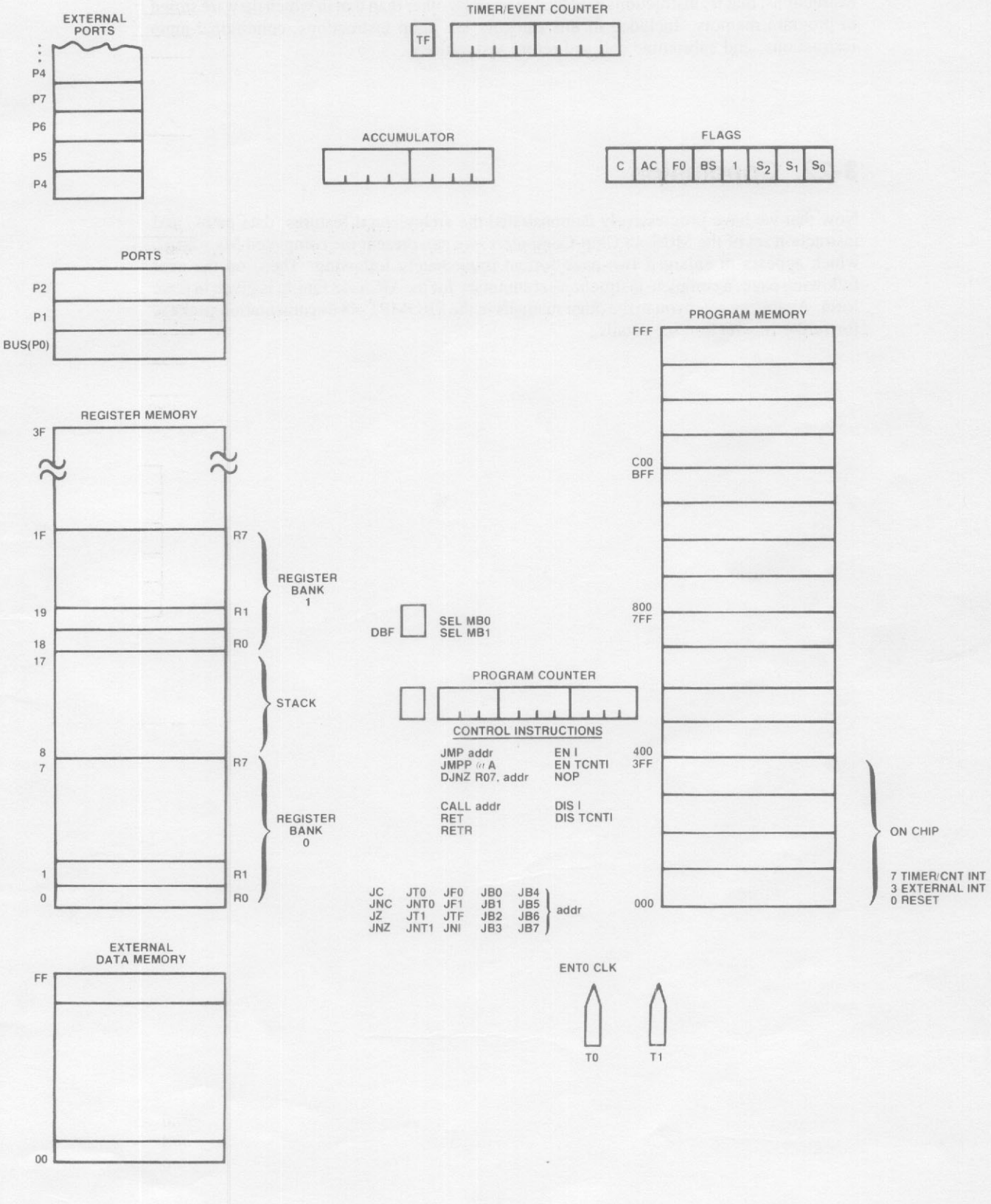
3-20. Input/Output Instructions

The Input/Output Instructions are those which transfer data between the accumulator and an I/O Port, or which in some way affect either the port or the data transferred through it. The Timer/Event Counter is considered as a programmable I/O device which generates an interrupt or which sets a flag when full, and whose contents are transferrable to the accumulator.


EXTERNAL PORTS
P4
P7
P6
P5
P4
TIMER/EVENT COUNTER
TF
ACCUMULATOR
ACCUMULATOR
FLAGS
C
AC
F0
BS
1
S_2
S_1
S_0
INC A
DEC A
CLR A
CPL A
DA A
SWAP A
RL A
RLCA
RRA
RRC A
PORTS
P2
P1
BUS(P0)
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
REGISTER MEMORY
3F
1F
19
18
17
8
7
1
0
R7
R1
R0
REGISTER BANK 1
STACK
REGISTER BANK 0
PROGRAM COUNTER
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET
EXTERNAL DATA MEMORY
FF
00


EXTERNAL PORTS
P4
P7
P6
P5
P4
TIMER/EVENT COUNTER
TF
ACCUMULATOR
MOV A, PSW
MOV PSW, A
FLAGS
C AC F0 BS 1 S2 S1 S0
CRC C CLR F0 SEL RB0
CPL C CPL F0 SEL RB1
CLR F1
CPL F1
F1
REGISTER ACCUMULATOR INSTRUCTIONS
PORTS
P2
P1
BUS(P0)
DIRECT
INDIRECT
IMMEDIATE
INC R07 @R01
DEC R07
DJNZ R07, addr
MOV A, R07 A, @R01 A, =
MOV R07, A @R01, A R07, =
@R01 =
XCH A, R07 A, @R01
XCHD A, @R01
ADD A, R07 A, @R01 A, =
ADC A, R07 A, @R01 A, =
ANL A, R07 A, @R01 A, =
ORL A, R07 A, @R01 A, =
XRL A, R07 A, @R01 A, =
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
MOVP A, @A
MOVP3 A, @A
REGISTER MEMORY
3F
1F
19
18
17
8
7
1
0
R7
R1
R0
REGISTER BANK 1
STACK
REGISTER BANK 0
PROGRAM COUNTER
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET
EXTERNAL DATA MEMORY
FF
00
MOVX A, @R01
MOVX @R01, A


EXTERNAL PORTS
P4
P7
P6
P5
P4
INPUT/OUTPUT INSTRUCTIONS
MOVD A, P47
MOVD P47, A
ANLD P47, A
ORLD P47, A
TIMER/EVENT COUNTER
TF
MOV A, T
MOV T, A
STRT T
STRT CNT
STOP TCNT
ACCUMULATOR
FLAGS
C AC F0 BS 1 S_2 S_1 S_0
IN(S) A, P12BUS
OUTL P12BUS, A
ANL P12BUS, =
ORL P12BUS, =
PORTS
P2
P1
BUS(P0)
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
REGISTER MEMORY
3F
1F
19
18
17
8
7
1
0
R7
R1
R0
REGISTER BANK 1
STACK
REGISTER BANK 0
PROGRAM COUNTER
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET
EXTERNAL DATA MEMORY
FF
00

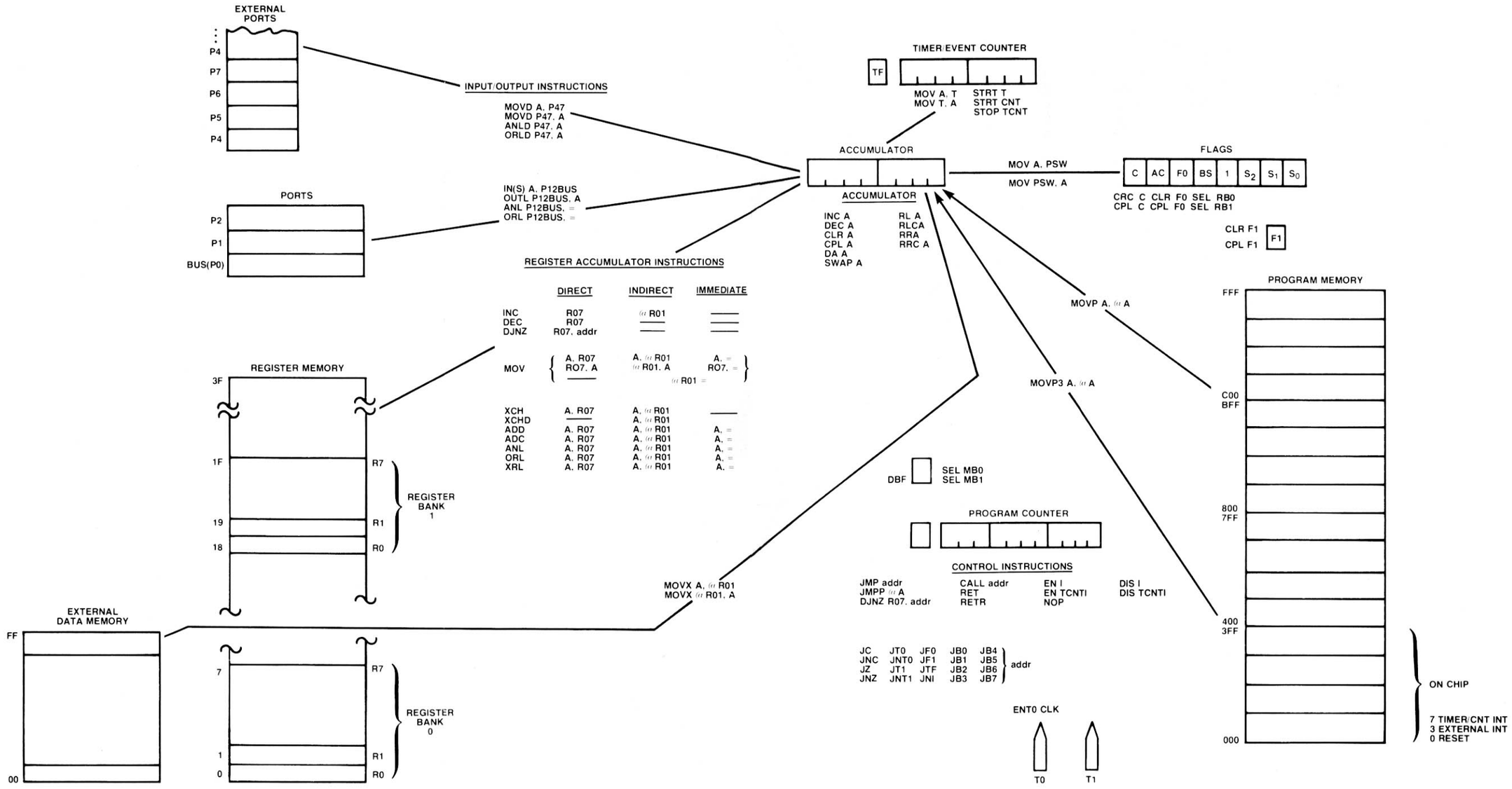

EXTERNAL PORTS
P4
P7
P6
P5
P4
TIMER/EVENT COUNTER
TF
ACCUMULATOR
FLAGS
C AC F0 BS 1 S_2 S_1 S_0
PORTS
P2
P1
BUS(P0)
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
REGISTER MEMORY
3F
1F
19
18
17
8
7
1
0
R7
R1
R0
REGISTER BANK 1
STACK
REGISTER BANK 0
DBF
SEL MB0
SEL MB1
PROGRAM COUNTER
CONTROL INSTRUCTIONS
JMP addr
JMPP @A
DJNZ R07, addr
EN I
EN TCNTI
NOP
CALL addr
RET
RETR
DIS I
DIS TCNTI
JC JT0 JF0 JB0 JB4
JNC JNT0 JF1 JB1 JB5
JZ JT1 JTF JB2 JB6
JNZ JNT1 JNI JB3 JB7
addr
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET
EXTERNAL DATA MEMORY
FF
00
ENT0 CLK
T0
T1

3-21. Control Instructions

Control Instructions are those instructions which allow the execution of non-sequential instructions; that is, instructions executed in an order other than that in which they are stored in program memory. Included in this category are jump instructions, conditional-jump instructions, and subroutine call and return instructions.

3-22. Conclusion

Now that we have progressively demonstrated the architectural features, data paths, and instruction set of the MCS-48 Chip-Computers, we can present the completed Micromap, which appears in enlarged two-page format immediately following. Then, on the next following page, a complete instruction set summary for the MCS-48 family is given in table form. Again, we refer you to the other manuals in the PROMPT-48 documentation package for further instruction set details.


EXTERNAL PORTS
P4
P7
P6
P5
P4
INPUT/OUTPUT INSTRUCTIONS
MOVD A, P47
MOVD P47, A
ANLD P47, A
ORLD P47, A
PORTS
P2
P1
BUS(P0)
IN(S) A, P12BUS
OUTL P12BUS, A
ANL P12BUS, =
ORL P12BUS, =
REGISTER ACCUMULATOR INSTRUCTIONS
DIRECT
INDIRECT
IMMEDIATE
INC R07 @R01
DEC R07
DJNZ R07, addr
MOV A, R07 A, @R01 A, =
R07, A @R01, A R07, =
@R01 =
XCH A, R07 A, @R01
XCHD A, @R01
ADD A, R07 A, @R01 A, =
ADC A, R07 A, @R01 A, =
ANL A, R07 A, @R01 A, =
ORL A, R07 A, @R01 A, =
XRL A, R07 A, @R01 A, =
REGISTER MEMORY
3F
1F
19
18
R7
R1
R0
REGISTER BANK 1
EXTERNAL DATA MEMORY
FF
00
7
1
0
R7
R1
R0
REGISTER BANK 0
MOVX A, @R01
MOVX @R01, A
TIMER/EVENT COUNTER
TF
MOV A, T
MOV T, A
STRT T
STRT CNT
STOP TCNT
ACCUMULATOR
MOV A, PSW
MOV PSW, A
FLAGS
C AC F0 BS 1 S2 S1 S0
CRC C CLR F0 SEL RB0
CPL C CPL F0 SEL RB1
ACCUMULATOR
INC A
DEC A
CLR A
CPL A
DA A
SWAP A
RL A
RLCA
RRA
RRC A
CLR F1
CPL F1
F1
PROGRAM MEMORY
FFF
C00
BFF
800
7FF
400
3FF
000
MOVP A, @A
MOVP3 A, @A
DBF
SEL MB0
SEL MB1
PROGRAM COUNTER
CONTROL INSTRUCTIONS
JMP addr
JMPP @A
DJNZ R07, addr
CALL addr
RET
RETR
EN I
EN TCNTI
NOP
DIS I
DIS TCNTI
JC JT0 JF0 JB0 JB4
JNC JNT0 JF1 JB1 JB5
JZ JT1 JTF JB2 JB6
JNZ JNT1 JNI JB3 JB7
addr
ENT0 CLK
T0
T1
ON CHIP
7 TIMER/CNT INT
3 EXTERNAL INT
0 RESET

CHAPTER 4
HOW THE PROMPT 48 WORKS

4-1. Introduction

As a complete low cost microcomputer design aid, the Prompt 48 requires many more features, both hardware and software, beyond the MCS-48 Chip-Computer itself, which the user mounts in an external Execution Socket. Besides the 8748 or 8035 execution processor, the Prompt box contains:

- 27-key front panel for Data/Control input
- An eight-character, 7-segment LED display (results/status out)
- Power supply
- 1k byte writable Program Memory (used in place of on-chip EPROM)
- 256 bytes of processor-external Data Memory
- An EPROM programmer, with external Programming Socket
- Bus and Port expansion capability for additional user memory or peripheral devices
- A serial port allowing for interface to an external terminal (TTY or RS-232)
- Hardware and Monitor firmware (4k bytes) to provide such services as Examine/Modify of Registers or Memory, and real-time execution of user programs with Single Step and Breakpoints.

The hardware features of the Prompt 48 are shown in functional block diagram form in Figure 4-1.

A few of the full capabilities of the MCS-48 Chip-Computer are restricted in the Prompt environment. This is due to design tradeoffs necessary to provide the full versatility of Prompt's features and functions. It is possible to work around these restrictions, which disappear once the development cycle is complete and the user system stands and runs alone, provided that you are aware of them in advance. In the course of the development in this chapter, they will be pointed out when appropriate, and summarized in Paragraph 4-9.

4-2. Hardware Description

All Prompt 48 circuitry is located on a single circuit board mounted on the inside of the front panel. A functional block diagram of this circuitry is given in Figure 4-1.

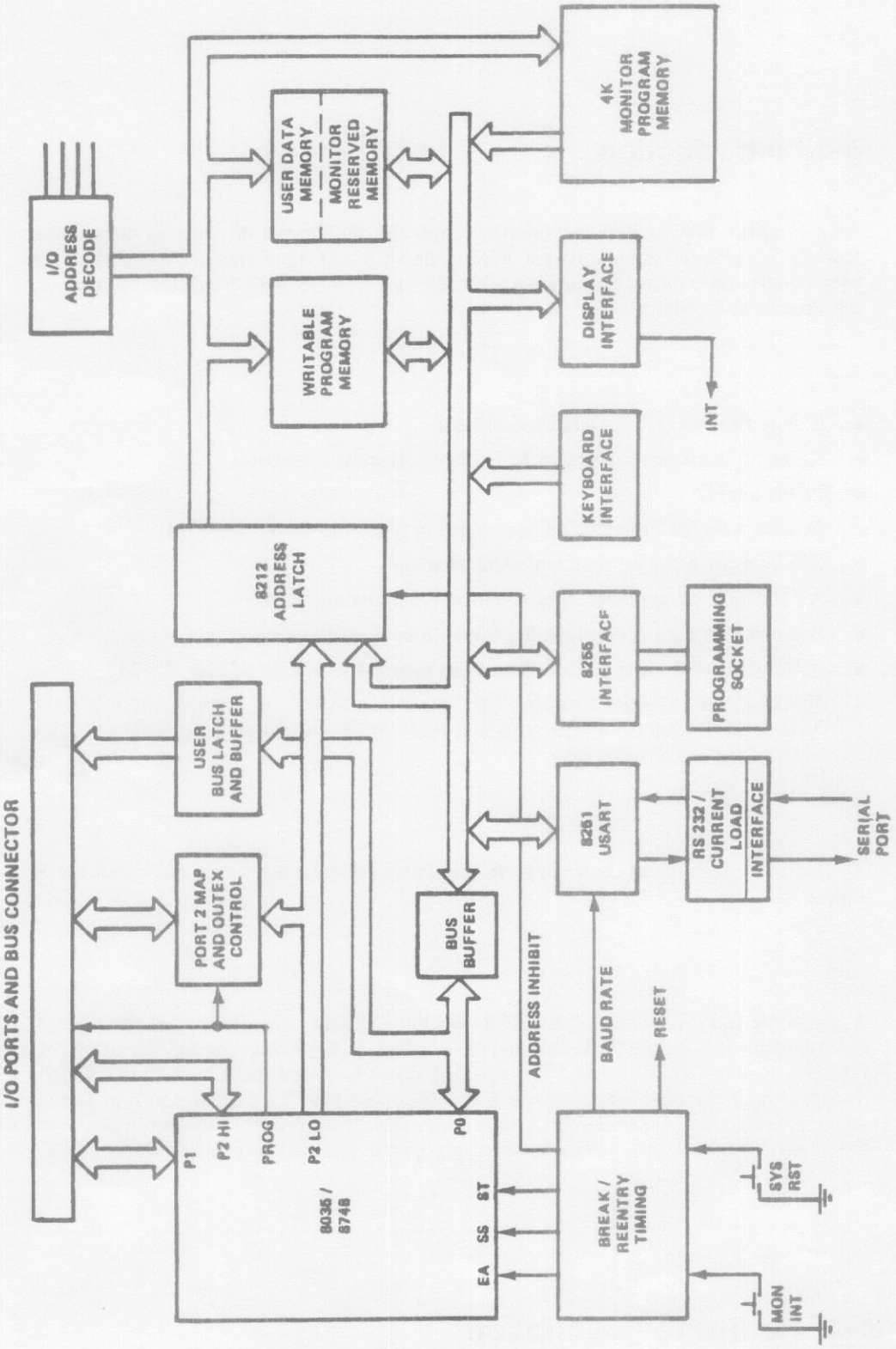


Figure 4-1. Functional Block Diagram

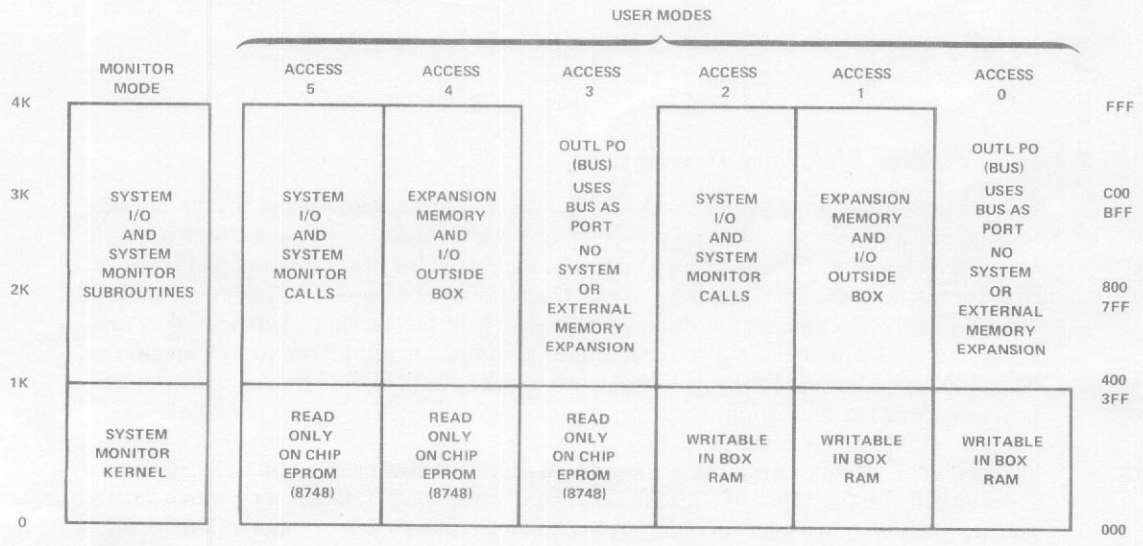
4-3. Memory

The memory in Prompt 48 consists of five different types:

a. 1k bytes read-only Program Memory internal to the 8748 or 8048 Chip-Computer
b. 4k bytes Monitor Firmware (program Memory, read-only)
c. 1k bytes writable Program Memory
d. 256 bytes User Data Memory
e. 256 bytes Monitor Data (Scratchpad) Memory

4-4. Program Memory

In the list above, the first three physical memories are program store memory, amounting to a total of 6k bytes. The MCS-48 Chip-Computer has a total addressing range in Program Memory of 4k bytes (12 address bits). The user can expand the writable Program Memory of the Prompt (item 3 above) up to the 4k limit by configuring his own external hardware via the Bus Connector (J1) and flat ribbon cable. If this were done, the Prompt ultimately would have to arbitrate Program Memory requests across a total range of 9k bytes, with a CPU whose address range is only 4k. This is accomplished indirectly in Prompt through "access codes." The user has at his disposal six access codes which he can enter through the appropriate Command Function on the keyboard (see Paragraph 5-13). For systems equipped with the 6MHz upgrade package, there are twelve access codes, including the six originals; refer to appendix J. Besides the user's access codes, the Monitor can map Program Memory in still another way. These seven access modes are summarized in the diagram below.



4-5. Data Memory

Prompt 48 has 256 bytes of internal Data Memory locations, not including the 64 on-chip Register Memory locations, available to the user as "External Data Memory," via the MOVX instructions.

External Data Memory can be examined and modified from the panel controls and displays, through the resources of the Monitor; the address range is from 0 to FF_{16}. In software, however, this Data Memory cannot be operated upon directly, like the working registers, or indirectly, like the remainder of on-chip Register Memory. To be operated upon, data from External Data Memory must first be moved into the accumulator by the use of the Move External Data Memory (MOVX A, @R01) command. This command does make use of indirect addressing via any of the four RAM pointer registers.

Similar to the Program Memory, the Data Memory space allotment is controlled by user selection in the user mode, and by hardware/firmware selection in Monitor mode. As a result, the user may select via the Access Code, whether the memory space above $3FF_{16}$ is to be expansion Data Memory or Monitor I/O functions. When neither expansion Data Memory or I/O is selected, the user need not be concerned with any address space above $3FF_{16}$. But if it is selected, a page addressing scheme above $3FF_{16}$ must be used, with Port 2 LSNibble used to select page number.

4-6. Input/Output

All I/O pins of the Execution Socket processor (MCS-48) are accessible via the I/O Ports and Bus Connector (J1) on the front panel of the Prompt, allowing the user to take full advantage of the Input/Output power of the MCS-48 Chip-Computer. But a great deal of I/O capability is already resident to Prompt as delivered: the full range of Command Functions described in Chapter 5 on Panel Operation are provided as inputs to the system by the firmware Monitor, together with the corresponding display outputs of status and data.

There is also a serial I/O option for Prompt, allowing communication with the system via a Teletype or RS-232C terminal. The installation of this option is described in Paragraph 6-14.

4-7. Monitor Firmware Description

The Prompt 48 System Monitor resides in 4k of non-volatile memory and is automatically activated by a bootstrap routine on power-up or System Reset. The Monitor is responsible for all maintenance of keyboard and display, and provides the full range of Command Functions as described in Chapter 5, "Panel Operation." A complete source code listing is included in the documentation package provided with the Prompt. This listing is self-documenting, including a rigorous structure definition of each Command Function in Backus-Normal Form. However, to make use of the powers of the Monitor, it is unnecessary to understand the listing.

Included in the Monitor firmware are a series of routines known as System Calls. These are general utility routines such as "read the keyboard" and "display character" which you the user are likely to find useful. In order to prevent unnecessary "re-invention of the wheel," these System Calls are made available to the user, and described in Appendix B. Note that Access Code 2 or 5 must be selected in order to access the Monitor memory where the System Calls reside.

4-8. Bus Expansion

In order to allow the user to expand the standard capabilities of the Prompt, some bus expandability has been included in the design. Bus expansion allows the addition of more Program Memory, more Data Memory, and the use of the 8243 I/O expansion chip. With a few exceptions, all bus lines and control signals are present on the J1 connector on the front panel of the Prompt (labled Bus Connector; a pin list for this connector is given in Table 4-1.) The lines *not* provided include EA (External Address), SS (Single Step), X1 and X2 (clock inputs), and there is a limitation on the bidirectionality of the LSNibble of Port 2. Due to the multipurpose nature of the LSNibble of Port 2 and the Bus, care must be exercised when interfacing to these ports to insure that the Access Code, P2 map, and external circuitry do not allow the Prompt interface drivers to compete with the user's drivers. (See Paragraph 6-14 for details.) In all cases, the user must instruct the Prompt as to the configuration of the system, including what type of expansion is desired. Since the EA line is not available, all user external Program Memory must reside above 1k. For hardware reasons, externally mapped Data Memory must be above 1k as well, though the External Data Memory provided by Prompt may be used from 0-FF_{16}. (LSNibble of Port 2 is used for mapping user-added external Data Memory.)

Table 4-1. Pin List for I/O Ports and Bus Connector

Signal Name	Pin No.	Buffer Characteristic
BUS (0)	17	3-STATE BIDIRECTIONAL
(1)	21	
(2)	25	
(3)	29	
(4)	31	
(5)	27	
(6)	23	
BUS (7)	19	
PORT 1 (0)	18	8748 PSEUDO BIDIRECTIONAL CHIP (NO BUFFER)
(1)	20	
(2)	22	
(3)	24	
(4)	26	
(5)	28	
(6)	30	
(7)	32	
PORT 2 (0)	7	3-STATE MAPPED BIDIRECTIONAL with 100 Ω IN SERIES
(1)	5	
(2)	3	
(3)	1	
PORT 2 (4)	4	8748 PSEUDO BIDIRECTIONAL CHIP (NO BUFFER)
(5)	6	
(6)	8	
(7)	10	
+ALE	13	TTL OUTPUT (10
+T0	14	CHIP BIDIRECTIONAL (CLOCK), 2.2K Pullup
+T1	12	CHIP INPUT, 2.2K input
−INT	49	1 TTL LOAD (MON. GATED)
−PSEN	15	TTL OUTPUT (10 LS LOADS)
−RD	9	
−WR	11	
−P0 WRITE	33	TTL OUTPUT (5 LS LOADS)
−PROG	2	CHIP OUTPUT (NO BUFFER)
−RESET	16	CHIP INPUT/OUTPUT (SYS RESET OVERRIDES), 2.2K pullup
GND	45, 46 47, 48	Ground

4-9. Restrictions

A few of the full capabilities of the MCS-48 Chip-Computer are restricted in the Prompt environment. This is due to design tradeoffs necessary to provide the full versatility of Prompt's features and functions. It is possible to work around these restrictions, which disappear once the development cycle is complete and the user system stands and runs alone, provided that you are aware of them in advance.

Monitor Reentry Uses Stack: When the MON INT key is pressed, the Monitor program interrupts the user program, using one stack entry. If the user has calculated his stack needs only for his own subroutines and interrupts, and has stored other data on the next available stack location, that data will be "zapped" (overwritten) by the user program return address.

Unsupported Instructions: ANL BUS, A and ORL BUS, A will not work except in Access Mode 3 and then with the GO/NO BREAK command. OUTL BUS, A can only be used in Access Modes 0 and 3.

Monitor Reentry Code: The upper 16 bytes of the lower 1k block of program Memory (addresses $3F0_{16}$ through $3FF_{16}$) must be reserved for the Prompt 48 Monitor reentry code. This code is automatically placed in Program Memory by the [7] Program PROM command. (See Paragraph 5-50.) These bytes must also be reserved when using the RAM Program Memory inside Prompt 48.

Access Code, P2 Map, LSN P2 Relationship: Care must be taken to insure that these three things are in agreement, as described in Paragraphs 5-13, 5-15, and 6-14.

Timer Routines: The Timer Interrupt is disabled when using the GO/WITH BREAK and GO/SINGLE STEP commands. To debug timer routines, insert JTF (Jump if Timer Flag = 1) in the program loop.

CHAPTER 5
PANEL OPERATION

5-1. Panel Description

The panel of the Prompt-48 provides the means for the user to communicate with the computer. Commands and data are entered through keys on the panel, and status and data are displayed through panel indicators. Figure 5-1 shows the layout of the Prompt-48 panel. It is divided into two functional control groups, and also has two 40-pin sockets: one a programming socket used for programming 8748s or 8741s, and the other an execution socket which holds the 8035, 8048 or 8748 processor which functions as the system's controller. In addition, there is a 50-pin flat cable connector which gives access to the executing processor's input/output ports and bus for a user prototype circuit.

5-2. Command Function Group

5-3. Display. The display device on the Prompt-48 consists of 8 seven-segment LED digits, together with LED periods between digits. These 8 digits are used to display hexadecimal information in three fields: Function (2 hex digits), Address (3 hex digits), and Data (3 hex digits). The system monitor (the program which reads user information input through the keys and displays information to be output in the LEDs) displays information

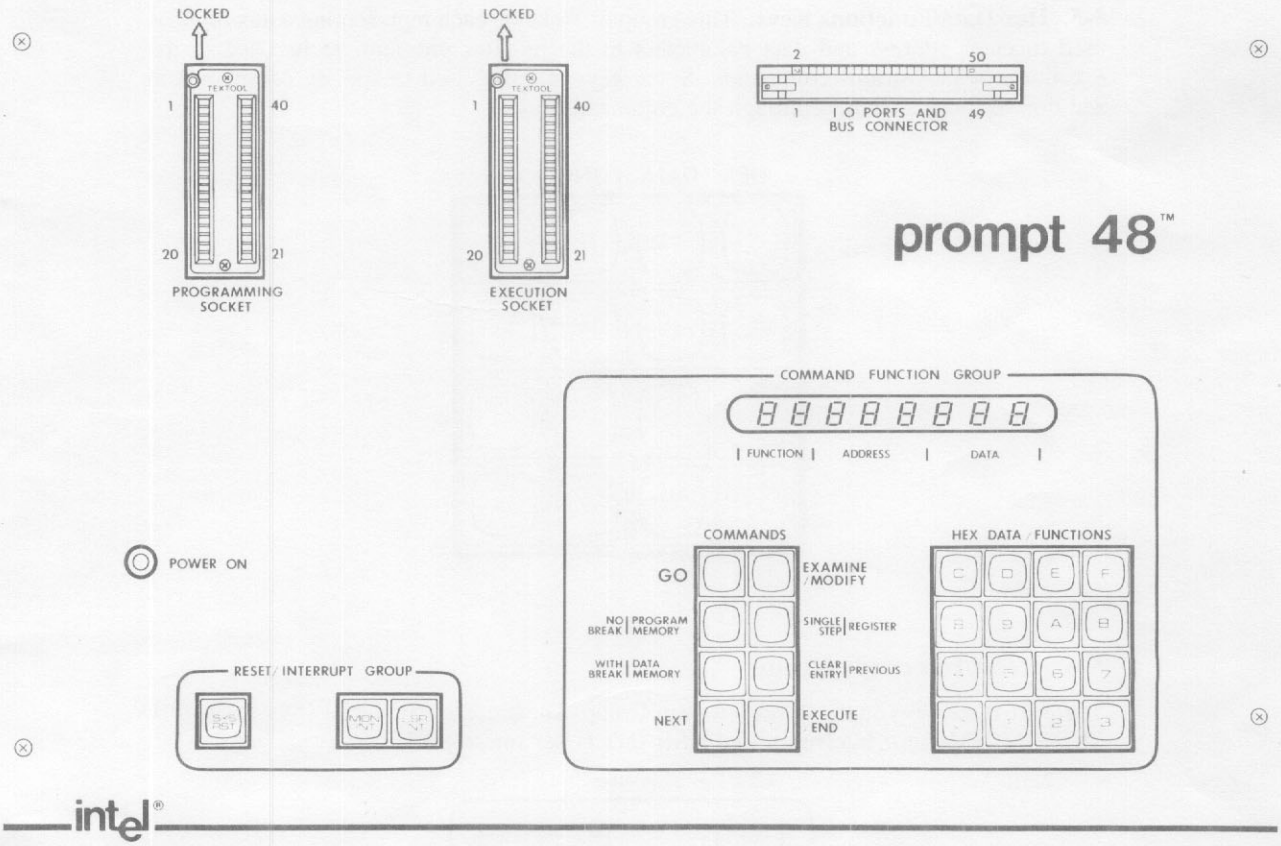
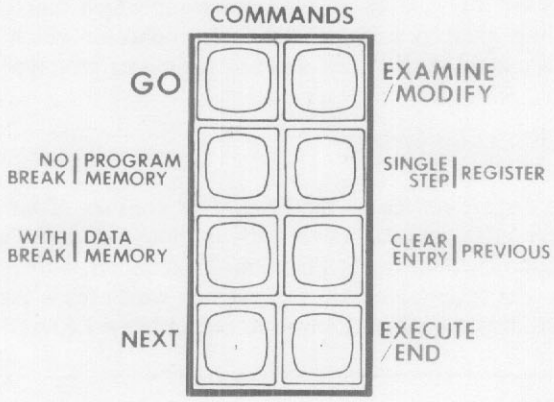


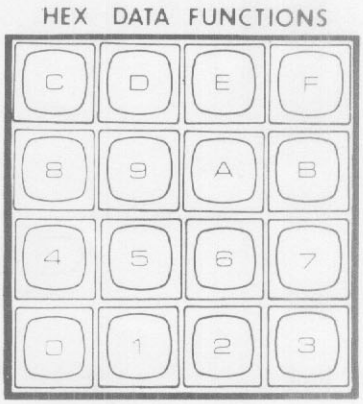
Figure 5-1. Prompt 48 Panel Layout

relevant to the current command being executed in these three fields. The LEDs are also available to user programs as output devices through the use of subroutines contained in the system monitor.

5-4. Command Keys. The keys in this eight-key group are used by the user to enter commands to the monitor program. Entering a command causes the monitor to display a 2 digit command code in the function field of the display.

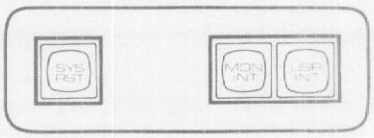


5-5. Hex Data/Functions Keys. This group of 16 keys, each representing a hex digit, is used to enter address and data parameters to the monitor program, to be used in the execution of the various commands. Some keys are also used to specify commands in addition to those specified through the commands keys.



5-6. Reset/Interrupt Group

There are three keys on the Reset/Interrupt Group. These are the SYS RST (System Reset), MON INT (Monitor Interrupt), and USR INT (User Interrupt) keys.



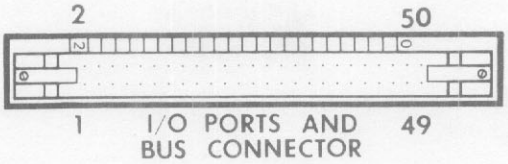
The SYS RST (System Reset) key is used to reinitialize the system hardware, reset the Access Mode to 0, and give control to the Prompt-48 monitor program. After actuating the SYS RST key, the ACCESS = 0 prompt should appear in the LED display.

The MON INT (Monitor Interrupt) key is used to interrupt the current process (user program) and turn over control to the monitor program so that its various functions are available. The interrupted user program can be continued later, as the user program status is saved by the Monitor program.

The USR INT (User Interrupt) key causes the Prompt-48 CPU to save its current program address and status on the stack and begin execution at program memory location 003, if interrupts have previously been enabled with the EN I instruction.

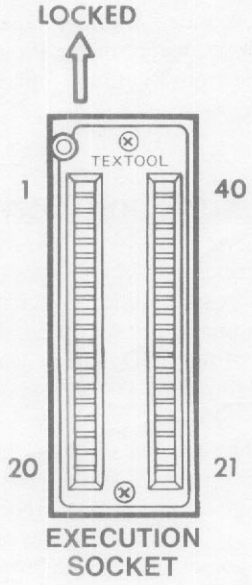
5-7. I/O Ports and Bus Connector (J1)

The I/O Ports and Bus Connector (J1) is provided to allow Prompt to exchange data with your external prototype device. It allows expansion of the Prompt-48 program memory, data memory, and I/O ports to the full capacity of the MCS-48 family. Details of hardware expansion through the I/O Ports and Bus Connector are given in Paragraph 6-14.



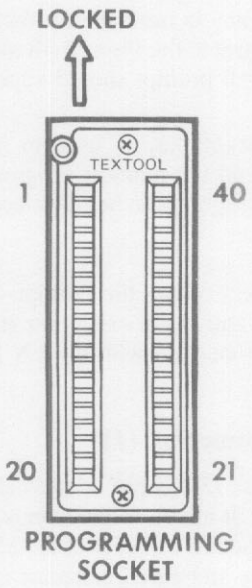
5-8. Execution Socket

The Execution Socket is a 40-pin zero insertion force socket in which resides the 8748, 8035, or 8048 chip-computer used to control the Prompt-48. The CPU chip in this socket runs the monitor and user programs specified by the user.



5-9. Programming Socket

The Programming Socket is a 40-pin zero insertion force socket which is used to program the 1k bytes of EPROM program memory in an 8748. It can also be used to program an 8741, or with an adapter (Prompt-475) to program an 8755. Another use of the Programming Socket is to read or verify the contents of any of these EPROM devices.



CAUTION

8048's should not be used in the Programming Socket as it is designed for use with EPROMS only.

CAUTION

MOS devices such as these are sensitive to transients or static electricity. It is possible to destroy their circuits by careless handling, especially if you are working in a carpeted area or in extremely low humidity conditions. Keep MOS devices in their protective packages when not in use. It is a good idea to touch the grounded frame of the computer with your hand before you place the EPROM device in the Programming Socket. This is to keep the pins of the device from being the first to touch and thereby absorb any static charges on your body.

5-10. Command Description Formats

In the following sections is described the various operating features of the Prompt-48 and how to use them. These features are accessible through the use of monitor commands. Each command is described with a command key sequence, those keys which must be pressed to call up the command, the state of the LED display when the command is specified, and ranges for all the parameters required by the individual command.

The command key sequence is the sequence of keys which must be pressed starting from a command prompt, and continuing through to the next command prompt (see Paragraph 5-12). The keys are indicated with square brackets: [NEXT] stands for the key in the lower left corner of the Commands keys. Key indications with capital letters, such as [GO], [PREVIOUS], or [D], stand for actual keys on the Prompt-48 panel. Key indications with lower case letters stand for command input parameters: [data] would stand for some element of information needed by the command, and input through the Hex Data keys.

Keys with multiple names appear in key indications using the name best fitting the context in which they appear. For example [EXECUTE], [END], and [.] all stand for the same key, but since this key is used in slightly different ways at different times, multiple names are used.

The command description sections conclude with a short example of the appropriate command.

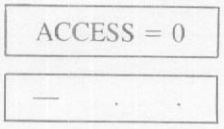
5-11. Command Input Options

The PROMPT Monitor is capable of accepting any of the commands from either of two sources: the keyboard, or the serial port. Following power-up, both devices are polled. The first one to send an input will be assigned as console until the next [SYS RST]. An input is defined as any keystroke for the keyboard, and as a non-null character for the serial port. The first serial character will be discarded while the keyboard first character will be accepted. When the serial port is selected, only handshaking signals are transmitted by PROMPT. These include a prompting character ["-"] to request each byte of data at monitor level, a character request ["?"] to request each byte of input data, and an error flag ["e"] if any command or character is unacceptable. Otherwide, data may be requested of the system by the standard dump command, which will output to the serial port in the usual manner (and in HEX record format).

It is important to note that the Monitor is looking for Hexidecimal, not ASCII codes. For this reason a Teletype keyboard, which generates ASCII coded data, is not really an effective substitute for the Prompt panel as an input device for commands. For example, the Fetch command is implemented by hitting an "F" on the Hex keypad of the Prompt. Inputting an "F" on the Teletype keyboard would result in a completely different code which the Monitor would not recognize. The usual reason for connecting a Teletype through the serial port would be to use it as a storage device (paper tape) and a hardcopy device (DUMP Program Memory, etc.).

5-12. Command Prompts

The command prompts are displayed in the command function group display to indicate to the user that the Prompt-48 monitor program is ready to accept a command. There are two command prompts:



The first prompt (ACCESS = 0) is given only when power is turned on or when the system is reset by pressing the SYS RST key. The second prompt (a dash on the display) is given subsequently to indicate the completion of a command and the system's readiness for the next command.

5-13. Access Mode Control

The Access Mode defines the configuration of the various memory and input/output features of the Prompt-48. The proper setting of the Access Mode is therefore critical to the operation of the Prompt-48.

Two things are specified by the Access Mode: which program memory is to be used, and how the Bus input/output port (port 0) is used. There is, in addition to the 1k bytes of program memory on the MCS-48 Chip-Computer, an additional 1k bytes of RAM memory in Prompt-48. This memory can be used in place of the 1k bytes of on-chip program memory for purposes of easy program development and modification. When using an 8035 in the execution socket, this is the only program memory available within Prompt-48. The Bus I/O port can be used in three ways:

a. As a port, latched on output. Under this mode OUTL BUS,A will work. However, ANL BUS,#data and ORL BUS,#data are not supported by Prompt-48 (refer to Paragraph 4-9.);

b. As a bus, to address expansion memory and I/O ports outside the Prompt-48 box; or

c. As a bus, to address the Prompt system monitor memory and I/O devices rather than any external hardware. This mode would be used if your user program wanted to talk directly to the Prompt keyboard, displays or serial channel. A listing of the system monitor program is included with your Prompt-48, and the use of some of its routines is described in Appendix B: System Calls.

5-14. Access Mode Select Command. The format of the Access Mode Select Command is as follows:

Command Key Sequence: [A] [data] [.]*

Function Display: "Ac. . 00"

Data Range: 0-5

Table 5-1. Summary Table of Access Mode Codes

Code	Program Memory	Bus Option
0	RAM	See Paragraph 5-13a
1	RAM	See Paragraph 5-13b
2	RAM	See Paragraph 5-13c
3	On-chip ROM/ EPROM	See Paragraph 5-13a
4	On-chip ROM/ EPROM	See Paragraph 5-13b
5	On-chip ROM/ EPROM	See Paragraph 5-13c

*EXECUTE/END key.

Example: Set Access = 0. The key sequence is [A] [O] [.]. Alternately, [SYS RST] sets Access = 0, as well as resetting various other system parameters.

The access codes are presented in a different format in Table 5-2.

Table 5-2. Access Code/P2 Map Summary

Access Code	Program Memory	System I/O & Calls	Expansion Memory & I/O	OUTL Port 0	Allowed LSN P2 Map
0	RAM	No	No	Yes	output (0) only
1	RAM	No	Yes	No	input or output
2	RAM	Yes	No	No	output only
3	On-chip ROM/EPROM	No	No	Yes	input or output
4	On-chip ROM/EPROM	No	Yes	No	input or output
5	On-chip ROM/EPROM	Yes	No	No	output only

5-15. Port 2 and Port 2 Mapping

In an MCS-48 Chip-Computer, the Least Significant Nibble (LSN) of Port 2 (P2) is used for a variety of functions. It is at various times an Input/Output port, a Data Memory page select, the Most Significant Nibble (MSN) of the Program Memory address, or some combination of these. In the case of Prompt-48, the monitor must be able to use the memory expansion capabilities, and yet at the same time allow the user to specify input/output, etc. To accomplish this, the port must be buffered. But in order to buffer, the direction of buffering must be specified. This is accomplished with the *P2 Map*.

The P2 Map is therefore nothing more than a bit-by-bit specification of the buffer direction of the corresponding bits of P2, with 1 = Input, and 0 = Output.

As mentioned above, MCS-48 Chip-Computers use the LSN P2 to address off-chip (expansion) Program Memory and I/O ports. The Access Code (see Paragraph 3-13) specifies the configuration and location of the various expansion memories and ports. Thus, in Prompt-48, the LSN P2 Map, the Access Code, and the contents of LSN P2 are all related. Furthermore, under some Access Codes, certain LSN P2 Maps could cause conflicts, and the Chip-Computer would not work! Be sure to carefully study the following information and the table which summarizes it.

With Access Codes 0, 2, or 5, LSN P2 Map must be output (0). In these modes LSN P2 is used by the monitor program to select various internal memories in the Prompt-48 and must not be affected by input devices.

Access Codes 1 or 4 allow LSN P2 Map to be either input or output. In these modes, the user program selects various external memories, I/O devices, and/or external ports which the user may have connected to the I/O Ports and Bus Connector, J1. The P2 Map is bypassed in these modes and therefore immaterial.

Access Code 3 also allows LSN P2 Map to be either input or output. Expansion memory and I/O ports are not allowed in this mode, and both P2 and Bus (P0) are available as I/O ports through J1.

This information is summarized in table 5-3, which also appears as Appendix I:

Table 5-3. Access Code/LSN P2 Map Summary

Access Code	Program Memory	System I/O & Calls	Expansion Memory & I/O	OUTL Port 0	Allowed LSN P2 Map
0	RAM	No	No	Yes	output (0) only
1	RAM	No	Yes	No	input or output
2	RAM	Yes	No	No	output only
3	On-chip ROM/EPROM	No	No	Yes	input or output
4	On-chip ROM/EPROM	No	Yes	No	input or output
5	On-chip ROM/EPROM	Yes	No	No	output only

5-16. Port 2 Map Command. The format of the Port 2 Map Command is as follows:

Command Key Sequence: [2] [data] [.]

Function Display: "P2. .MM"

Data Range: MM_{16} where MM are two hexadecimal digits chosen to map the eight lines of P2 according to table 5-4.

Table 5-4. Port 2 Map Command Data Bits Vs. Port 2 Pin Numbers

MS Nibble				LS Nibble			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin P27	Pin P26	Pin P25	Pin P24	Pin P23	Pin P22	Pin P21	Pin P20

A hexadecimal/binary conversion is given in table 5-5. 0 = Output, 1 = Input.

Table 5-5. Hexadecimal/Binary Conversion

Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Example: Set P2 Map = 00_{16} (all lines of P2 mapped as outputs). The key sequence is 2.].

It should be noted that Port 2 is treated by the monitor as "register" 47 and can be examined and/or modified through the Examine/Modify Register command (see Paragraph 5-17).

5-17. Examine/Modify Commands

5-18. Examine/Modify Program Memory Command. The format of the Examine/Modify Program Memory Command is as follows:

Command Key Sequence: [EXAMINE/MODIFY] [PROGRAM MEMORY] [address] [NEXT] [data] [NEXT] [data] . . . [.]

Function Display: "EP. . "

Address Range: $0\text{-}FFF_{16}$

Data Range: $0\text{-}FF_{16}$

The Examine/Modify Program Memory command is used to examine and/or modify one or more Program Memory locations. An address in Program Memory is specified, and optional data is input if desired to replace the existing data displayed in the DATA field of the LEDs. The next greater address in Program Memory can be examined by then pressing the [NEXT] key, or the next lesser address can be examined by pressing the [PREVIOUS] key. Return to command prompt is accomplished by pressing the [END] key instead of another [NEXT] or [PREVIOUS]. The Program Memory accessed through this command is the RAM Program Memory in Prompt-48 and expansion Program Memory the user may have connected to J1. To read the on-chip EPROM Program Memory of an 8748 or 8741, the EPROM contents must first be read into RAM with the Prom Fetch command (see Paragraph 5-50).

Example: Change Program Memory locations 3A and 3B to contain 5C and E2, respectively. The key sequence is [EXAMINE/MODIFY] [PROGRAM MEMORY] [3] [A] [NEXT] [5] [C] [NEXT] [E] [2] [.]. This could also be accomplished by pressing [EXAMINE/MODIFY] [PROGRAM MEMORY] [3] [B] [NEXT] [E] [2] [PREVIOUS] [5] [C] [.], or by individually modifying locations 3A and 3B in separate command sequences.

5-19. Examine/Modify Register Command. The format of the Examine/Modify Register Command is as follows:

Command Key Sequence: [EXAMINE/MODIFY] [REGISTER] [address] [NEXT] [data] [NEXT] [data] . . . [.]

Function Display: "Er. "

Address Range: $0\text{-}48_{16}$

Data Range: $0\text{-}FF_{16}$

This command allows the user to examine and optionally modify the 64 bytes of Register Memory on-chip with MCS-48 Chip-Computers. As with the other Examine/Modify commands, [PREVIOUS] may be substituted for any [NEXT] after the first to examine the previous register contents, or [.] may be substituted to terminate the command sequence.

There are in Prompt-48 an additional 9 bytes of special-purpose "Register" memory, in address locations $40_{16}\text{-}48_{16}$. These "Register" Memory locations represent other registers in the Chip-Computer, such as the Accumulator, etc. according to table 5-6.

Table 5-6. Special Purpose Register Memory Summary

Register Address	Significance
40	Accumulator
41	Timer/Event Counter
42	Flags Register → CY | AC | F0 | BS | F1 | STP →
43	Program Counter Low Byte
44	Program Counter High Byte
45	Bus (Port 0)
46	Port 1
47	Port 2
48	Prompt-48 Misc. → Cntr Run | Timr Run | Timr Flag | Int Nest | Int Enab | Mem Bank | TEST 1 | TEST 2

Ports 0 and 1 (''registers'' 45 and 46) cannot be modified by the Examine/Modify Register command. They are read only.

The bits of Prompt-48 Misc. (''register'' 48) require some explanation:

COUNTER RUN must be set to ''1'' if your program uses the MCS-48 timer/event counter as an event counter. This allows the monitor to suspend and restart the timer/event counter when a break in the user program occurs.

During breaks the Prompt-48 monitor saves the state of the broken user program so that it can be restored as execution is resumed.

TIMER RUN will be set to ''1'' on break if the timer is running. If you clear this bit to ''0'' the timer will not be restarted when execution is resumed.

TIMER FLAG allows you manually to examine and modify the user timer flag.

NESTED FROM INTERRUPT will be set to ''1'' if you have broken during a routine servicing an interrupt. This is a user state bit.

WILL ENABLE INTERRUPT represents the user's interrupt enable state if user interrupts are enabled.

MEM BANK is the Designated Bank Flag (refer to paragraph 3-8).

T1 and T0 are the MCS-48 test inputs and are read only.

Example: Change the contents of Register Memory location 2A to be 49_{16}. The key sequence is [EXAMINE/MODIFY] [REGISTER] [2] [A] [NEXT] [4] [9] [.].

5-20. Go Commands and Breakpoints

5-21. Go/No Break Command. The format of the Go/No Break Command is as follows:

Command Key Sequence: [GO] [NO BREAK] [address] [.]

Function Display: "Go. . "

Address Range: 0-FFF$_{16}$

The Go/No Break command causes the MCS-48 Chip-Computer in the Execution Socket to begin program execution at the address in Program Memory given in the command sequence. Program execution will continue until either (1) control is returned to the monitor by pressing [MON INT], or (2) the system is reset and control given to the monitor by pressing [SYS RST]. The CPU runs at full speed.

Example: Begin execution of a program in PROGRAM Memory which starts at 1F0$_{16}$. The key sequence is [GO] [NO BREAK] [1] [F] [0] [.].

5-22. Breakpoints. A breakpoint is a location in program memory which, when reached by the user program, causes control to be given to a monitor program. The state of the processor is saved so that the current user program can be continued at a later time. Control is then given to the monitor program so that the user can examine register contents, memory contents, and so forth as an aid to program development and debugging.

The Prompt-48 monitor allows the user to specify up to eight breakpoints, numbered 0-7. When running with breakpoints enabled (using the Go/With Break command) the monitor single-steps the user program and checks after each step to see if a breakpoint address has been reached in Program Memory. If it has, the monitor program suspends stepping, saving the contents of all the MCS-48 registers, and displays information about which breakpoint was reached, the contents of the Program Counter, and the contents of the Accumulator. The monitor then allows the user access to all of the panel commands. If no other keys have been pressed, the user program may be restarted by pressing [NEXT]. If other keys have been pressed, one of the Go commands must be used.

These breakpoints do not affect memory contents. They may even be set in non-writable ROM or PROM.

5-23. Examine/Modify Breakpoint Command. The format of the Examine/Modify Breakpoint Command is as follows:

Command Key Sequence: [B] [breakpoint number] [NEXT]
[breakpoint address] [NEXT]
[breakpoint address] [NEXT]
. . . [.]

Function Display: "br. . "

Breakpoint Number Range: 0-7 (Appears in ADDRESS display field)

Breakpoint Address Range: 0-FFF$_{16}$ (Appears in DATA display field)

The Examine/Modify Breakpoint command operates in a manner similar to the Examine/ Modify Program Memory, Data Memory, and Register commands. In this case the address is the breakpoint number, and the data is the location in Program Memory where the

Clr Brk [B] [0] [END] Br. 0. - - -

breakpoint resides. As with the other Examine/Modify commands [PREVIOUS] can be substituted for any [NEXT] after the first, or [END] can be substituted to terminate the command sequence.

Example: Set Breakpoints 0 and 1 at Program Memory locations 106_{16} and $3F2_{16}$, respectively. The key sequence is [B] [0] [NEXT] [1] [0] [6] [NEXT] [3] [F] [2] [.].

5-24. Go/With Break Command. The format of the Go/With Break Command is as follows:

Command Key Sequence: [GO] [WITH BREAK] [address] [.]

Function Display: "Gb. . "

Address Range: $0\text{-}FFF_{16}$

The Go/With Break command single steps the MCS-48 Chip-Computer through the user program starting at the address in Program Memory given in the command sequence. Program single stepping will continue until either (1) [SYS RST] is pressed, (2) [MON INT] is pressed, or (3) a breakpoint is reached. Breakpoint information is displayed in the format,

"bN.ADR. AC",

where N = the breakpoint number, ADR = the contents of the Program Counter (the breakpoint address), and AC = the contents of the Accumulator. The monitor then allows the user access to all of the panel commands. If no other keys have been pressed, the user program may be restarted by pressing [NEXT]. If other keys have been pressed, one of the Go commands must be used.

Example: Begin execution of a program in Program Memory which starts at $E0_{16}$, with breakpoints enabled. The key sequence is [GO] [WITH BREAK] [E] [0] [.].

5-25. Search Memory Commands

The Search Memory commands allow the user to search Program Memory, Data Memory, or Register Memory for a one- or two-byte data pattern, called the search target. The commands which search for a one-byte search target are called Byte Search commands, and those which search for a two-byte search target are called Word Search commands.

The format for each of the Search Memory commands is the same, as follows:

[search type] [memory type] [starting address]
[NEXT] [ending address] [NEXT] [search target]
[NEXT] [mask] [EXECUTE] [NEXT] [NEXT] . . . [.],

where [search type] is [4] for a Byte Search or [5] for a Word Search; [memory type] is [PROGRAM MEMORY], [DATA MEMORY], or [REGISTER]; [starting address] and [ending address] define the area to be searched; [search target] is the object of the search; and [mask] is a bit pattern the same length as [search target], which causes only those bits in [search target] which correspond to 1's in [mask] to be tested in the search. The sequence [NEXT] [mask] is optional and may be omitted.

The [EXECUTE] key causes the search to commence. If no occurrences of the search target (as modified by the mask) are found in the specified memory range, the monitor returns to command prompt status. If the (modified) search target is found, the address of the occurrence and the data matching the (modified) search mask are displayed as follows:

"SM.ADR. DD",

where M is the memory type, ADRis the address in hexadecimal of the occurrence, and DD is the data matching the (modified) search target. After [EXECUTE] is pressed and data is found, [NEXT] may be pressed to reinitiate the search with ADR+1 as the new starting address. All other search parameters remain constant.

5-26. Byte Search Program Memory Command. The format of the Byte Search Program Memory Command is as follows:

Command Key Sequence: [4] [PROGRAM MEMORY] [starting address]
[NEXT] [ending address] [NEXT] [search target]
[NEXT] [mask] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "SP. . "

Address Range: 0-FFF_{16}

Search Target Range: 0-FF_{16}

Mask Range: 0-FF_{16}

Note: [NEXT] [mask] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Program Memory for the second occurrence of $6C_{16}$ or $6D_{16}$ (01101100_2 or 01101101_2) between the addresses 100_{16} and $2D0_{16}$. This implies a mask of FE_{16} (11111110_2). The key sequence is [4][PROGRAM MEMORY][1][0][0] [NEXT][2][D][0][NEXT][6][C][NEXT][F][E][EXECUTE][NEXT][.].

5-27. Byte Search Data Memory Command. The format of the Byte Search Data Memory Command is as follows:

Command Key Sequence: [4] [DATA MEMORY] [starting address]
[NEXT] [ending address] [NEXT] [search target]
[NEXT] [mask] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "Sd. . "

Address Range: 0-FF_{16}

Search Target Range: 0-FF_{16}

Mask Range: 0-FF_{16}

Note: [NEXT] [mask] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Data Memory between 00 and $4B_{16}$ for the first occurrence of $A9_{16}$. The key sequence is [4] [DATA MEMORY] [0] [NEXT] [4] [B] [NEXT] [A] [9] [EXECUTE] [.].

5-28. Byte Search Register Memory Command. The format of the Byte Search Register Memory Command is as follows:

Command Key Sequence: [4] [REGISTER] [starting address]
[NEXT] [ending address] [NEXT] [search target]
[NEXT] [mask] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "Sr. . "

Address Range: $0\text{-}48_{16}$

Search Target Range: $0\text{-}FF_{16}$

Mask Range: $0\text{-}FF_{16}$

Note: [NEXT] [mask] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Register Memory for the first occurrence of $8X_{16}$, where X signifies "don't care". This implies a mask of $F0_{16}$ (11110000_2). The key sequence is [4] [REGISTER] [0] [NEXT] [4] [8] [NEXT] [8] [0 or any other hex key] [NEXT] [F] [0] [EXECUTE] [.].

5-29. Word Search Program Memory Command. The format of the Word Search Program Memory Command is as follows:

Command Key Sequence: [5] [PROGRAM MEMORY] [starting address]
[NEXT] [ending address]
[NEXT] [search target MSB]
[NEXT] [search target LSB]
[NEXT] [mask MSB] [NEXT] [mask LSB] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "SP. . "

Address Range: $0\text{-}FFF_{16}$

Search Target Range: $0\text{-}FFFF_{16}$

Mask Range: $0\text{-}FFFF_{16}$

Note: [NEXT] [mask MSB] [NEXT] [mask LSB] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Program Memory for the first occurrence of $A429_{16}$. The key sequence is [5] [PROGRAM MEMORY] [0] [NEXT] [F] [F] [F] [NEXT] [A] [4] [NEXT] [2] [9] [EXECUTE] [.].

5-30. Word Search Data Memory Command. The format of the Word Search Data Memory Command is as follows:

Command Key Sequence: [5] [DATA MEMORY] [starting address]
[NEXT] [ending address]
[NEXT] [search target MSB]
[NEXT] [search target LSB]
[NEXT] [mask MSB] [NEXT] [mask LSB] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "Sd. . "

Address Range: 0-FF_{16}

Search Target Range: 0-$FFFF_{16}$

Mask Range: 0-$FFFF_{16}$

Note: [NEXT] [mask MSB] [NEXT] [mask LSB] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Data Memory between locations 19_{16} and $3F_{16}$ for the first occurrence of $3B14_{16}$. The key sequence is [5] [DATA MEMORY] [1] [9] [NEXT] [3] [F] [NEXT] [3] [B] [NEXT] [1] [4] [EXECUTE] [.].

5-31. Word Search Register Memory Command. The format of the Word Search Register Memory Command is as follows:

Command Key Sequence: [5] [REGISTER] [starting address]
[NEXT] [ending address]
[NEXT] [search target MSB]
[NEXT] [search target LSB]
[NEXT] [mask MSB] [NEXT] [mask LSB] [EXECUTE]
[NEXT] [NEXT] . . . [.]

Function Display: "Sr. . "

Address Range: 0-48_{16}

Search Target Range: 0-$FFFF_{16}$

Mask Range: 0-$FFFF_{16}$

Note: [NEXT] [mask MSB] [NEXT] [mask LSB] and [NEXT] [NEXT] . . . are optional and may be omitted.

Example: Search Register Memory for an occurrence of $A42D_{16}$. The key sequence is [5] [REGISTER] [0] [NEXT] [4] [8] [NEXT] [A] [4] [NEXT] [2] [D] [EXECUTE] [.].

5-32. Move Memory Commands

The Move Memory commands allow the user to move blocks of data from one area to another in any one of the three memory types: Program Memory, Data Memory, or Register Memory. Data cannot be moved from one memory type to another.

The format for each of the Move Memory commands is the same, as follows:

[9] [memory type]
[source starting address] [NEXT] [source ending address]
[NEXT] [destination starting address] [EXECUTE]

where [9] is the Move Memory command; [memory type] is [PROGRAM MEMORY], [DATA MEMORY], or [REGISTER]; [source starting address] and [source ending address] define the block of data to be moved; and [destination starting address] defines the area of memory to which the data is to be moved.

The memory move is commenced by pressing [EXECUTE]. The Move Memory commands will move any block of memory data between any two memory areas of a single memory type.

5-33. Move Program Memory Command. The format of the Move Program Memory Command is as follows:

Command Key Sequence: [9] [PROGRAM MEMORY]
[source starting address] [NEXT]
[source ending address] [NEXT]
[destination starting address] [EXECUTE]

Function Display: "nP. . "

Address Range: $0\text{-}FFF_{16}$

Example: Move the contents of Program Memory locations $0\text{-}FF_{16}$ to Program Memory locations $270_{16}-26F_{16}$. The key sequence is [9] [PROGRAM MEMORY] [0] [NEXT] [F] [F] [NEXT] [2] [7] [0] [EXECUTE] [.].

5-34. Move Data Memory Command. The format of the Move Data Memory Command is as follows:

Command Key Sequence: [9] [DATA MEMORY]
[source starting address] [NEXT]
[source ending address] [NEXT]
[destination starting address] [EXECUTE]

Function Display: "nD. . "

Address Range: $0\text{-}FF_{16}$

Example: Move the contents of Data Memory locations $6\text{-}2B_{16}$ to Data Memory locations $A_{16}-2F_{16}$ (move the block "up" in memory four bytes). The key sequence is [9] [DATA MEMORY] [6] [NEXT] [2] [B] [NEXT] [A] [EXECUTE] [.].

5-35. Move Register Memory Command. The format of the Move Register Memory Command is as follows:

Command Key Sequence: [9] [REGISTER]
[source starting address] [NEXT]
[source ending address] [NEXT]
[destination starting address] [EXECUTE]

Function Display: "nr. . "

Address Range: $0\text{-}48_{16}$

Example: Move the contents of Register Memory locations $4\text{-}18_{16}$ to Register memory locations $A_{16}-1E_{16}$. The key sequence is [9] [REGISTER] [4] [NEXT] [1] [8] [NEXT] [A] [EXECUTE] [.].

5-36. Clear Memory Commands

5-37. Clear Program Memory Command. The format of the Clear Program Memory command is as follows:

Command Key Sequence: [C] [PROGRAM MEMORY] [starting address] [NEXT] [ending address] [.]

Function Display: "CP. . "

Address Range: 0-FFF_{16}

The Clear Program Memory command clears each memory location between and including the starting address and the ending address to 00_{16}.

Example: Clear Program Memory locations 0-$3FF_{16}$. The key sequence is [C] [PROGRAM MEMORY] [0] [NEXT] [3] [F] [.].

5-38. Clear Data Memory Command. The format of the Clear Data Memory Command is as follows:

Command Key Sequence: [C] [DATA MEMORY] [starting address] [NEXT] [ending address] [.]

Function Display: "Cd. . "

Address Range: 0-FF_{16}

The Clear Data Memory command clears each memory location between and including the starting address and the ending address to 00_{16}.

Example: Clear Data Memory locations $20_{16}-4F_{16}$. The key sequence is [C] [DATA MEMORY] [2] [0] [NEXT] [4] [F] [.].

5-39. Clear Register Memory Command. The format of the Clear Register Memory Command is as follows:

Command Key Sequence: [C] [DATA MEMORY] [starting address] [NEXT] [ending address] [.]

Function Display: "Cr. . "

Address Range: 0-48_{16}

The Clear Register Memory command clears each memory location between and including the starting address and the ending address to 00_{16}.

Example: Clear Register Memory locations 0-$1F_{16}$. The key sequence is [C] [REGISTER] [0] [NEXT] [1] [F] [.].

5-40. Dump Memory Commands

Any data that can be accessed by the Examine/Modify Memory commands may be output through the serial port with these commands. The user may thereby save program, register, or data information on paper tape, or hard copy or other peripheral. The format is the hexadecimal Object File format, described in Appendix D.

If the command is received from the Prompt-48 keyboard, then the Hexadecimal Object File will be preceded and followed by a series of null characters for tape header and trailer. If the command is received via the serial channel, then the Hexadecimal Object File will be immediately dumped without any null insertion.

5-41. Dump Program Memory Command. The format of the Dump Program Memory Command is as follows:

Command Key Sequence: [D] [PROGRAM MEMORY] [starting address] [NEXT] [ending address] [.]

Function Display: "dP. . "

Address Range: 0-FFF_{16}

Note: starting address must be less than or equal to ending address.

Example: Dump Program Memory locations 0-$3FF_{16}$ through the serial port. The key sequence is [D] [PROGRAM MEMORY] [0] [NEXT] [3] [F] [F] [.].

5-42. Dump Data Memory Command: The format of the Dump Data Memory Command is as follows:

Command Key Sequence: [D] [DATA MEMORY] [starting address] [NEXT] [ending address] [.]

Function Display: "dd. . "

Address Range: 0-FF_{16}

Note: starting address must be less than or equal to ending address.

Example: Dump Data Memory through the serial port. The key sequence is [D] [DATA MEMORY] [0] [NEXT] [F] [F] [.].

5-43. Dump Register Memory Command. The format of the Dump Register Memory Command is as follows:

Command Key Sequence: [D] [REGISTER] [starting address] [NEXT] [ending address] [.]

Function Display: "dr. . "

Address Range: 0-48_{16}

Note: starting address must be less than or equal to ending address.

Example: Dump Register Memory through the serial port. The key sequence is [D] [REGISTER] [0] [NEXT] [4] [8] [.].

5-44. Enter Into Memory Commands

The Enter into Memory commands allow the user to load any file corresponding to the Hexadecimal Object File Format (Appendix D) from the serial port into Program Memory, Data Memory, or Register Memory. The parameters needed by the Enter commands are the memory type and an offset to the starting address given in the Object File.

5-45. Enter into Program Memory Command. The format of the Enter into Program Memory command is as follows:

Command Key Sequence: [E] [PROGRAM MEMORY] [starting address offset] [.]

Function Display: ''rP. . ''

5-46. Enter into Data Memory Command. The format of the Enter into Data Memory command is as follows:

Command Key Sequence: [E] [DATA MEMORY] [starting address offset] [.]

Function Display: ''rd. . ''

5-47. Enter into Register Memory Command. The format of the Enter into Register Memory command is as follows:

Command Key Sequence: [E] [REGISTER] [starting address offset] [.]

Function Display: ''rr. . ''

5-48. Hexadecimal Arithmetic Command

5-49. Hexadecimal Arithmetic Command. The format of the Hexadecimal Arithmetic command is as follows:

Command Key Sequence: [6] [x data] [NEXT] [y data] [EXECUTE]

Function Display: ''HE. . ''

Data Range: $0\text{-}FFF_{16}$

The Hexadecimal Arithmetic command performs hexadecimal addition and subtraction on two one-to-three digit hexadecimal numbers, x and y. Upon pressing [EXECUTE] the sum and difference are displayed in the following format:

''HE.x+y.x−y''.

5-50. EPROM Programming, Fetch, Compare Commands

5-51. EPROM Programming Command. The EPROM Programming commands allow the user to program all or part of the EPROM Program Memory on an 8748 Microcomputer with EPROM, an 8755 EPROM Program Memory and I/O Expander, or an 8741 Microcomputer with EPROM (UPI-41 family).

There are two programming modes, one which does not insert the Prompt-48 byte reentry code, and one which does.

The mode which does insert this code is intended for 8748's which are to be used in the Prompt-48 Execution Socket. The Prompt-48 16 byte reentry code is needed in Program memory to allow the Monitor program to properly transfer control to the user program. It occupies the 16 highest bytes of the lower 1024 bytes of Program Memory, locations $3F0_{16}-3FF_{16}$. This programming mode is inappropriate for 8755's present via an adapter, and an error display will appear in the LED's to indicate that the wrong mode has been selected.

The programming mode which does not insert the reentry code copies the RAM Program Memory faithfully to the EPROM device in the Programming Socket. This mode will work for 8741's, 8748's, and with the addition of a Prompt 475 adapter, 8755's.

The Prompt-48 will not attempt to program EPROM devies which have not had the appropriate locations completely erased. If an unerased location is detected an error display with the address and EPROM contents will appear.

5-52. Program EPROM With Reentry Code Command. The format of the Program EPROM With Reentry Code command is as follows:

Command Key Sequence: [7] [starting address] [NEXT] [ending address] [NEXT] [starting EPROM address] [EXECUTE]

Function Display: "Pr 8748"

Address Range: $0-3FF_{16}$

This command programs all or part of the EPROM Program Memory on an 8748 Micocomputer with EPROM. The 16 byte monitor reentry code is automatically substituted for any of the 16 locations from $3F0_{16}$ to $3FF_{16}$.

Example: Program the Prompt-48 RAM Program Memory into an 8748 intended for use in the Prompt-48 Execution Socket. First install the 8748 in the Programming Socket. The key sequence is [7] [0] [NEXT] [3] [F] [F] [NEXT] [0] [EXECUTE]. The display will blank to indicate that the EPROM is being programmed and the command prompt returns automatically after the EPROM has been successfully programmed.

5-53. Program EPROM Without Reentry Code Command. The format of the Program EPROM Without Reentry Code command is as follows:

Command Key Sequence: [3] [starting address] [NEXT] [ending address] [NEXT] [starting EPROM address] [EXECUTE]

Function Display: "Pr 8741" or "Pr 8755"*

Address Range: $0-3FF_{16}$

*with 475 adapter.

This command programs all or part of the EPROM Program Memory on the 8741, 8748, or 8755 (if a Prompt 475 adapter is installed). The function display "Pr 8741" appears for the 8741 and 8748, and (if the adapter is installed) the function display "Pr 8755" appears. With this command the RAM Program Memory is written to the EPROM device without modification.

Example: Program the entire Prompt-48 RAM Program Memory contents into the EPROM device on an 8741. First install the 8741 into the Prompt-48 Programming Socket. The key sequence is [3] [0] [NEXT] [3] [F] [F] [NEXT] [0] [EXECUTE]. The LED display will blank to indicate that the EPROM is being programmed and the command prompt returns automatically after the EPROM has been successfully programmed.

5-54. Compare EPROM Command. The format of the Compare EPROM command is as follows:

Command Key Sequence: [8] [starting Prompt address] [NEXT]
[ending Prompt address] [NEXT]
[starting EPROM address] [EXECUTE]

Function Display: "Co. . " ??

Address Range: 0-FFF_{16} (but not to exceed PROM capacity)

The Compare EPROM command compares the specified areas of Prompt-48 RAM Program Memory and the EPROM device installed in the Programming Socket. Before specifying this command, an 8748, 8741, or 8755 with 475 adapter must be installed in the Programming Socket. If no EPROM device or 475 adapter is present and locked, or the device is placed in the socket backwards, upon receipt of the [C] command the display will read out an error message.

Example: Compare the contents of an 8748 installed in the Programming Socket with the RAM Program Memory in Prompt-48. The key sequence is [8] [0] [NEXT] [3] [F] [F] [NEXT] [0] [EXECUTE].

5-55. Fetch EPROM Command. The format of the Fetch EPROM command is as follows:

Command Key Sequence: [F] [starting Prompt address] [NEXT]
[ending Prompt address] [NEXT]
[starting EPROM address] [EXECUTE]

Function Display: "FP. . "

Address Range: 0-FFF_{16} (but not to exceed PROM capacity)

The Fetch EPROM command moves the contents of the EPROM Program Memory of the device installed in the Programming Socket to the RAM Program Memory in Prompt-48. Before specifying this command, an 8748, 8741, or 8755 with 475 adapter must be installed in the Programming Socket. If no EPROM device or 475 adapter is present and locked, or if the device is placed in the socket backwards, upon receipt of the [F] command the display will read out an error message.

Example: Read the contents of an 8748 installed in the Programming Socket into the RAM Program Memory in Prompt-48. The key sequence is [F] [0] [NEXT] [3] [F] [F] [NEXT] [0] [EXECUTE].

Table 5-7. Command List Summary

Command Prompts: "ACCESS=0" and "— . . "		
Command Key(s)/(Description)	**Function Display**	**Section**
[GO]:	"G . . "	5-20
– [NO BREAK]	"Go . . "	5-21
– [WITH BREAK]	"Gb . . "	5-24
– [SINGLE STEP]	"GS . . "	5-24
[EXAMINE/MODIFY]:	"E . . "	5-17
– [PROGRAM MEMORY]	"EP . . "	5-18
– [DATA MEMORY]	"Ed . . "	5-17
– [REGISTER]	"Er . . "	5-15
[2] (Port 2 Map)	"P2 . . MM"	5-16
[3] (Program PROM — 8741 or 8748)	"Pr 8741 "	5-53
[3] (Program PROM — 8755, with adapter)	"Pr 8755 "	5-53
[4] (Byte Search):	"S1 . . "	5-25
– [PROGRAM MEMORY]	"SP . . "	5-26
– [DATA MEMORY]	"Sd . . "	5-27
– [REGISTER]	"Sr . . "	5-28
[5] (Word Search):	"S2 . . "	5-25
– [PROGRAM MEMORY]	"SP . . "	5-28
– [DATA MEMORY]	"Sd . . "	5-30
– [REGISTER]	"Sr . . "	5-31
[6] (Hexadecimal Arithmetic)	"HE . . "	5-49
[7] (Program PROM — 8748)	"Pr 8748 "	5-52
[8] (Compare PROM)	"Co . . "	5-54
[9] (Move Memory):	"n . . "	5-32
– [PROGRAM MEMORY]	"nP . . "	5-33
– [DATA MEMORY]	"nd . . "	5-34
–. [REGISTER]	"nr . . "	5-35
[A] (Access Mode Select)	"Ac . . CC"	5-14
[B] (Examine/Modify Breakpoint)	"br . . "	5-23
[C] (Clear Memory):	"C . . "	5-36
– [PROGRAM MEMORY]	"CP . . "	5-37
– [DATA MEMORY]	"Cd . . "	5-38
– [REGISTER]	"Cr . . "	5-39
[D] (Dump Memory):	"d . . "	5-40
– [PROGRAM MEMORY]	"dP . . "	5-41
– [DATA MEMORY]	"dd . . "	5-42
– [REGISTER]	"dr . . "	5-43
[E] (Enter into Memory):	"r . . "	5-44
– [PROGRAM MEMORY]	"rP . . "	5-45
– [DATA MEMORY]	"rd . . "	5-46
– [REGISTER]	"rr . . "	5-47
[F] (Fetch PROM)	"FP "	5-55

CHAPTER 6
HOW TO USE PROMPT 48

6-1. Setting Up A System

As mentioned in the introductory chapter of this manual, your decision to use the Prompt-48 as a development system was likely based on the observation that software design and debug time is the critical path that stands between where you are now and a completely engineered product. The hardware aspects of system design using the MCS-48 family of components, though not trivial, are greatly simplified by the forethought and modularization of that family.

In this chapter we will refer to your prototype of the desired end product as the user system. This chapter will attempt to guide you in the efficient use of the development tools of the Prompt-48, while giving the briefest of coaching in the modern discipline of systems engineering.

6-2. Education

The first step is to become familiar with what the microcomputer is and what it can do. For this, unless you are already familiar with the subject, reference should be made to Chapter Three of this manual, "How the INTEL Chip-Computers Work." An extensive documentation package is included with Prompt-48, and this should also be consulted. In particular, you should become familiar with the contents of *MCS-48 Microcomputer User's Manual* and the *Prompt 48 Reference Cardlet*.

If time is critical in getting started in microprocessors, designers or managers can attend one of many INTEL-sponsored 3-day training courses which give basic instruction in the MCS-48 as well as hands-on experience with MCS-48 development systems.

After general familiarization is complete, either through self-instruction or a training course, the next step is to gain a better "feel" for what a microprocessor can do in your own applications by writing several exercise programs which perform basic functions. You may require such things as I/O routines for various sorts of ports; or delays, counting functions, look-up tables, arithmetic functions, and logical operations which can serve as a set of building blocks for future applications programs. Several basic programming examples are included in the Prompt-48 documentation package, such as the "Stopwatch" program described and listed in Appendix C of this manual. The Intel User's Library is a source of more specific applications routines.

6-3. Functional Definition

After a thorough grounding in the basics of microcomputing has been achieved, the functions of the intended user system should be thoroughly defined and documented. So many "correct" methods for this sort of documentation exist that it is impossible to make dogmatic prescriptions for all situations.

A traditional protocol of design-supportive documentation is the flowchart method. This familiar device, for which templates and other drafting aids exist, calls for a separate "black box" with summary description within for each distinct "function" to be performed by the computer; also, the proper sequencing and interconnection of functions, including the possibility that certain paths may only be remote options, seldom used.

We will employ a different discipline of program design in this chapter and in Appendix C, known as structured programming through Warnier-Orr diagrams. Rather than "graphics-oriented" like flowcharts, this documentation is analogous to indented outlines. Examples appear in Paragraph 6-6.

6-4. Hardware Configuration

The next step involves the definition of the microcomputer hardware necessary to implement the complete user system. In general, any system will include CPU (Central Processor), Program Memory and Data Memory, Input/Output, and the appropriate interfaces with the outside world. It will already be apparant that the MCS-48 component family answers many system-building questions in a straightforward manner. In the first place, the 8748, if selected as Central Processor, already includes the first one thousand bytes of Program Memory, the first 64 bytes of Register (data) Memory, and three 8-bit I/O ports. For those many applications requiring no more resources, the 8748 (or its masked ROM equivalent, the 8048) would have only a few hardware needs beyond the chip itself: a power supply (which could be a battery), a simple oscillator or clock, a minimal amount of interface/support circuitry, and possibly a chassis or other packaging.

But most user applications will be more involved than this, requiring a detailed hardware system design study and the use of other components in the MCS-48 family. Such a design study would require the separate consideration of requirements in Input/Output, Memory and Throughput. Input/Output and Memory will now be discussed, but Throughput will be covered in the subsection which follows, "Code Generation."

Input/Output capability must be defined in terms of number of inputs, number of outputs, bi-directional lines, latching or non-latching I/O, output drive capability, and input impedence.

In terms of Memory requirements, a separate study is necessary for Register (Data) Memory and for Program Memory. The number of words of RAM storage required for intermediate results and other data storage must be determined, and a decision made as to whether off-chip expansion is needed. (An additional 256 bytes can be directly added, and up to 4K bytes indirectly; see Paragraph 6-14 for details.) The type of system will dictate whether battery backup is needed to maintain data in RAM during power failure.

Probably the most difficult parameter to define initially is the amount of Program Memory needed to store the final user program. Although previously written exercise programs will make this estimate more accurate, a generous amount of "breathing room" should be allowed in program memory until coding is complete and the exact requirements are known. The Prompt-48 allows for 1k byte (one thousand bytes) of RAM memory for program development. If more proves to be necessary, the user can configure it externally to Prompt with the Bus Connector (J1) and flat cable. (MCS-48 has an upward address limit of 4k in Program Memory.)

The problem of "trade-offs" of hardware versus software is familiar to every experienced system designer. For example, many special functions such as serial data communications (TTY or RS-232) or keyboard/display interfaces may be implemented in software (programs); however, in cases where these functions place a severe load on the processor in terms of time or Program Memory, special peripheral interface circuits such as the 8251, Universal Synchronous or Asynchronous Receiver/Transmitter (USART) or 8279 Keyboard/Display interface may be used.

We are only sketching the essentials of hardware system development in this section. For full details, see Paragraph 6-14.

6-5. Code Generation

The writing of the final program code for the application can begin once the system function and hardware have been defined and can be generated in parallel with the detailed hardware design (PC card layout, power supply, etc.) Often the final hardware definition is not possible, however, until some or all of the coding is complete; the memory requirements, both for Program Memory and Data Memory, may be unpredictable. Also, it may not be possible to predict, in certain time-critical real-time applications, whether the processor will

have sufficient throughput. ''Benchmark'' programs, which are typically only the most critical sequences in a complete applications program, are often written in completely coded form for the purpose of more exactly predicting memory and throughput requirements.

Throughput is defined loosely as the ''amount of computing'' that a system can accomplish in a given time interval. Although a fast processor like the MCS-48 has throughput ''overkill'' for most applications, it is easy to conceive that a sufficiently challenging real-time application would overtax its processing power. For example, in some industrial control application, a feedback loop between ''sensing'' and ''correcting'' might need to be repeatedly established very quickly, say within 1/100th of a second. Such a final, dedicated applications program may be able, in addition to any general ''housekeeping'' or record-keeping duties, to periodically read the current outside-world data appearing at an input port; to perform data analysis calculations; to compute a feedback or correction factor; and to write this to an output port — all within perhaps 1/100th of a second.

If benchmark programs are carefully-selected and completely coded, it is possible to make literal and accurate calculations for the time required to execute them. One simply counts the number of bytes in the benchmark program (object code) and multiplies by the instruction cycle time of the MCS-48. Assuming a clock frequency of 3 MHz (3 million cycles per second), the basic instruction cycle for the fetching/executing of a program byte would be 5.0 microseconds long. (Reference the *MCS-48 Microcomputer User's Manual*.) Note that most MCS-48 instructions generate only one byte of object code, but that many have operands requiring a second byte.

The whole process of applications software development, from program design to final coding, is described in Paragraph 6-6.

6-6. Programming Techniques

The first part of this section is aimed primarily at beginning or intermediate assembly language programmers. While it is not sufficient as a general introduction to assembly language programming, it is intended to present concepts allowing efficient software development in the MCS-48 environment. The advanced programmer may wish to spend some time briefly examining the subsections on Program Design and Program Test and Debugging for interest's sake.

The subsections:

Assembling JMP and CALL Instructions,
Program Memory Paging, and
Prompt-48 Considerations,

are of general interest as they discuss aspects pertaining specifically to the MCS-48 family or Prompt 48.

The MCS-48 Assembly Language Programming Manual should be consulted as a detailed reference for all MCS-48 CPU software.

6-7. Program Design

The first step in the design of any system, hardware or software, is to define the problem. Only when the exact function of the application is determined can the resources necessary to execute that function be determined.

A common phrase in programming these days is ''top-down'' program design. By this we mean that the designer divides the problem into smaller separate sections to be solved separately. The words ''top-down'' describe the hierarchial or pyramid-like way in which this division is made. As an example, let's say that we are to design a program which will

allow a particular MCS-48 system to function as a stopwatch; perhaps we will design it to run on Prompt 48 itself. When we say "stopwatch," the precise instructions needed aren't immediately obvious. The problem must be divided into simpler sub-problems. One possible division might be into subsections called: Display Functions, Timer Control Functions, Data Functions, User Input/Output Functions, and so forth. These subsections of the program are then themselves divided and subdivided until the problem is reduced to a number of vastly simpler problems, such as adding 1 to the contents of a given memory location. The final set of simple problems is then solved one at a time, and called the program modules.

Figure 6-1 shows a possible partial breakdown of the stopwatch problem. While the figure shows the organizational structure of the program, it does not indicate how the modules communicate with one another.

The communication between modules is the second major phase of program design, called designing the modular interfaces, which are simply the ways in which modules pass control and data back and forth. For example, in the stopwatch the User Control Functions (Commands) module must give control to one of its submodules, whose task is to read the keyboard for user commands. The Read Keyboard For Command submodule would examine keys and return control to the calling module. It would also pass data back to the calling module indicating which key, if any, was pressed. The simpler these modular interfaces are kept the easier it is to assemble all modules into a working program. For this reason the modular breakdown process should attempt to separate the problem into sub-problems which depend as little as possible on each other for data.

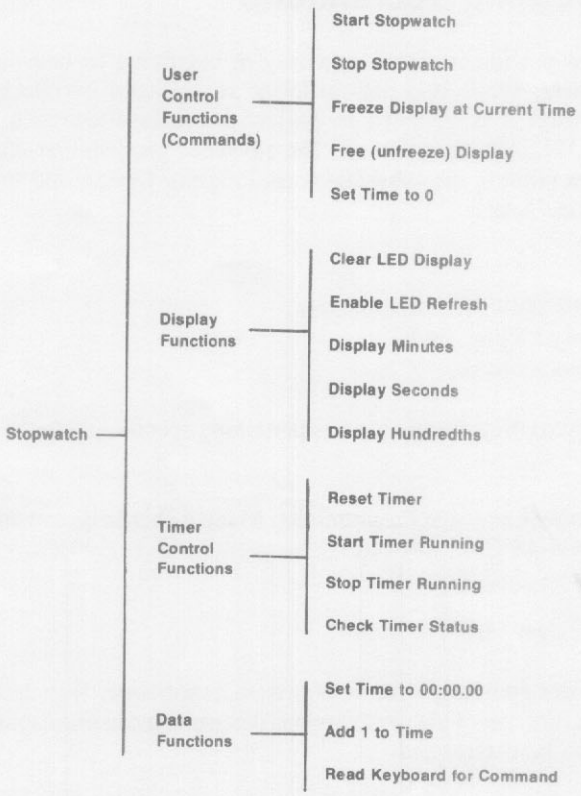


Figure 6-1. Stopwatch Program Structure

When a given task must be performed the same way in two or more modules, it can be made into a subroutine. A good example of a subroutine is a multiplication routine. The multiplication routine receives control, and the two numbers to be multipled, from the calling routine, multiplies the two numbers together, and returns control and the product to the calling routine. Since subroutines are called from a number of different areas in the program, the address of the caller must be saved in order for control and data to pass back to the calling module. This is accomplished very simply in the MCS-48 Chip-Computers and is described in Paragraph 3-9.

The concept of executive modules is also useful. Briefly, an executive module is any module which controls other modules as subroutines. This idea can be applied at any level in the structure of the program, just as the idea of modules itself. In the stopwatch structure, the User Control Functions module is executive to the other three modules on its level.

6-8. Hand Assembly

When each program module and modular interface has been specified, the individual modules must be translated into a form the computer can deal with. The first step of this translation is to write the program in assembly language according to the MCS-48 Assembly Language Manual. If an ISIS-II or other development system is available to then assemble the assembly language program into machine language, hand assembly need not be used; otherwise, the hexadecimal machine code contents of Program Memory must be determined manually.

Let's look at a simple example. Consider a single-module program which is to count. That is, it will repeatedly add 1 to a specified memory location. The task of the program (module) might look something like this:

1. Replace the variable COUNT with COUNT+1.
2. Repeat step 1.

The next step is to assign the location of the data called COUNT. Let's put it in Working Register 0. Now write the instructions:

```
START:  MOV A,#1      ;Put 1 in the Accumulator
        ADD A,R0      ;Add 1 to R0 (COUNT)
        MOV R0,A      ;Replace R0 with R0+1
        JMP START     ;Repeat forever.
```

The mnemonic instructions with comments are collectively called the program source code. The hexadecimal contents of Program Memory which the source code stands for are called the program object code. The essence of hand assembly is the translation from source code to object code.

In our example we must now do just that: assemble the program.

First, a starting address must be chosen; say Program Memory location 0. Write the address of each instruction at the extreme left of your program sheet:

Addr	*Label*	*Ins*	*Opnd*	*Comment*
000	START:	MOV	A,#1	;Put 1 in the Accumulator

Then go down the instructions one at a time, assigning hexadecimal values to the Program Memory address in the left column:

Addr	*Data*	*Label*	*Ins*	*Opnd*	*Comment*
001	2301	START:	MOV	A,#1	;Put 1 in the Accumulator
002	68		ADD	A,R0	;Add 1 to R0 (COUNT)
003	A8		MOV	R0,A	;Replace R0 with R0+1
004	0400		JMP	START	;Repeat forever.

The hexadecimal values can be looked up in the *Prompt-48 Reference Cardlet* (''by mnemonic'' section), the *MCS-48 Microcomputer User's Manual,* or the *MCS-48 Assembly Language Programming Manual.*

6-9. Program Test and Debugging

When each of the program modules and their modular interfaces have been identified and written into an assembly language program, some effort should be devoted to determining whether or not the program works as intended. This effort is called program testing. Removing the errors uncovered by program testing is called debugging.

Large programs are frequently far too complex to exhaustively test as a whole. One answer to this problem is to test as thoroughly as possible each module and modular interface individually. If this is done carefully, the programmer is almost certain to have a correctly working program when the modules are assembled, unless there are serious flaws in the overall program design structure. The bugs that do (almost inevitably) crop up can usually be identified as originating in a particular module and/or modular interface, and are easily fixed.

In order to evaluate the performance of an individual module, its communication process with other modules must be simulated. For example, if a multiplication routine is to be tested, the input data (the numbers to be multiplied) must be somehow provided, and the output data (product) must be available for verification. Thus, the stand-alone routine to be verified must be provided with an ''environment'': that is, it must be surrounded with sufficient other assembly-language instructions so that it can be run in the computer with simulated values. Such a test program would be called a ''dummy routine,'' and the practice of pre-verifying individual modules before the program is run as a whole is often referred to as ''echo checking.'' If the module is found to be faulty, the resources of the development system must be called on to trace its internal operation.

The basic facilities for testing modules and modular interfaces in Prompt-48 are the Go/With Break, Go/Single Step, and Examine/Modify commands.

Breakpoints allow the user to stop program execution at pre-planned points in order to supply input data, examine output data, check the status of various registers, and so forth. The placement of breakpoints and use of the Go/With Break command are discussed in Paragraph 5-20.

The Go/Single Step command allows the user to execute a routine instruction-by-instruction, verifying the routine's operation at each step. The use of this command is described in Paragraph 5-20.

The Examine/Modify commands are the means by which all this verification takes place. The MCS-48 registers and Data Memory are accessible through these commands, as shown in Paragraph 5-17.

6-10. Program Memory Paging

In MCS-48 Chip-Computers, Program Memory is divided into from 4 to 16 256-byte pages. There are only two ways for program execution to cross page boundaries: the use of the JMP or CALL instructions. The address of the next instruction to be executed is kept in the Program Counter. After most instructions, only the lower eight bits are modified to form the next address ($2^8 = 256$). With the JMP and CALL instructions, however, an additional three bits are included as part of the instruction. The twelfth bit of the Program Counter (BS) is also replaced by the DBF bit with the execution of these instructions. The JMP and CALL instructions are therefore the only instructions which can transfer control to anywhere in the $2^{12} = 4096$ bytes of Program Memory.

The DBF bit controls whether a JMP or CALL instruction passes control to a destination above or below the $2^{11} = 2048$ byte Program Memory boundary. This is accomplished by replacing bit 11 (the twelfth bit) of the Program Counter, BS, with DBF on a JMP or a CALL instruction. DBF is controlled with the SEL MB0 and SEL MB1 instructions. SEL MB0 replaces DBF with 0, and subsequent JMP or CALL instructions will have destination address of $0\text{-}7FF_{16}$. SEL MB1 replaces DBF with 1, and JMP's and CALL's will have destinations of $800_{16}-FFF_{16}$.

6-11. Assembling JMP and CALL Instructions

With the JMP and CALL instructions, three bits of the destination address (next Program Counter contents) are included in the hexadecimal object code for the particular instruction involved. These bits are Program Counter bits 10, 9, and 8. They specify any 256 byte page of Program memory in either of two Program Memory banks, $000\text{-}7FF_{16}$ or $800_{16}-FFF_{16}$. To determine which page of the given memory bank the destination lies in, take the full address ($000\text{-}FFF_{16}$) and subtract 800_{16} from any address which is 800_{16} or greater. The resultant page number indicates the proper JMP or CALL instruction's object code.

The precise manner in which the JMP and CALL instructions operate is discussed in Paragraphs 3-8 and 3-9, and the *MCS-48 Microcomputer User's Manual*.

6-12. Care and Feeding of EPROMS

At a certain point in program development you will make the decision that the process is complete: that is, you will have verified that the program works as designed. Hopefully you will already have attempted a certain number of dry runs under "dummy" parameters, in an attempt to force the program into some sort of fluke under extreme conditions; perhaps it will only be a random and arbitrary selection of parameters. Now it is time to commit the proven program to non-volatile EPROM, either the 1k resident on the 8748 processor, or possibly the 2k 8755 EPROM Program Memory and I/O Expander device.

To do this, carefully insert the chip in the Programming socket with the marked pin on the chip next to the numeral "1" on the Prompt's panel insuring proper orientation. There are numerous cautions to observe while doing so. In the first place, never insert a processor into the Programming socket unless a second processor (such as the 8035 provided with your Prompt) is properly locked in the Execution socket. Secondly, the chips are fragile! Dropping, twisting, or uneven pressure may break them. Also, avoid putting any pressure on the quartz window area of the processor. Finally, as MOS devices the EPROMs are subject to damage by static electricity contacting the pins. Never place the pins near any metallic surface except the Prompt socket itself; and even then, discharge any residual charges by touching your hand to the Prompt chassis before inserting the chip. At all other times, keep the chip safe in its protective foam cushion.

The final step in EPROM programming is to execute one of the instructions for this purpose detailed in Paragraph 5-50.

If for any reason it is desired to erase a programmed EPROM to allow for reprogramming, it is only necessary to expose it to light with wavelengths of light shorter than approximately 4000 Angstroms (ultraviolet). Sunlight and certain fluorescent lamps have wavelengths in the 3000 Å– 4000 Å range. If the 8748 is to be exposed to sunlight or room fluorescent lighting for extended periods, then opaque labels should be placed over the window, to prevent unintentional erasure.

The recommended erasure procedure is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (UV intensity multiplied by exposure time) for erasure should be a minimum of 15 W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/em^2 power rating. The 8748 should be placed within one inch from the lamp tube during exposure. Some lamps have a filter on their tube and this filter should be removed before erasure.

6-13. Prompt 48 Considerations

A few of the full capabilities of the MCS-48 Chip-Computer are restricted in the Prompt environment. This is due to design tradeoffs necessary to provide the full versatility of Prompt's features and functions. It is possible to work around these restrictions, which disappear once the development cycle is complete and the user system stands and runs alone, provided that you are aware of them in advance.

Monitor Reentry Uses Stack: When the MON INT key is pressed, the monitor program interrupts the user program, using one stack entry. If the user has calculated his stack needs only for his own subroutines and interrupts, and has stored other data on the next available stack location, that data will be "zapped" (overwritten) by the user program return address.

Unsupported Instructions: ANL BUS,A and ORL BUS, A will not work except in Access Mode 3 and then only with the GO/NO BREAK command. OUTL BUS,A can only be used in Access Modes 0 and 3.

Monitor Reentry Code: The upper 16 bytes of the lower 1k block of Program Memory (addresses $3F0_{16}$ through $3FF_{16}$) must be reserved for the Prompt 48 Monitor reentry code. This code is automatically placed in Program Memory by the [7] Program EPROM command. (See Paragraph 5-50.) These bytes must also be reserved when using the RAM Program Memory inside Prompt 48.

Access Code, P2 Map. LSN P2 Relationship: Care must be taken to insure that these three things are in agreement, as described in Paragraph 5-13, 5-15, and 6-14.

Timer Routines: The Timer Interrupt is disabled when using the GO/WITH BREAK and GO/SINGLE STEP commands. To debug timer routines, insert JTF (Jump if Timer Flag = 1) in the program loop.

6-14. Hardware Considerations

In expanding either Program or Data Memory, the first step is to define how the expanded memory is to be partitioned, i.e., Program vs. Data. In your final MCS-48 design, processor control signals will distinguish Program Memory accesses from Data memory accesses: PSEN/ signals instruction fetches from Program Memory, and RD/ and WR/ signal accesses to Data Memory. Thus your final design will be a "Aiken" machine, with separate Program and Data Memory (see Chapter 3).

However, during debugging you may find a "von Neumann" machine to be useful, particularly if you are expanding Program Memory. While checking out software you need to easily load and modify all of Program Memory, 1k or more. Expansion Program and Data

Memory may be joined into a single, all-RAM store—though Program and Data addresses may not overlap. In hardware, you must now AND the PSEN/ and RD/ signals to form what we'll call EXRD/ (expansion read). This signal selects the expansion RAM whenever an instruction is fetched from expansion Program Memory or when an expansion Data Memory location is read.

A design for such "von Neumann" expansion memory is shown in figure 6-2. Table 6-1 is a list of the pin functions on the I/O Ports and Bus Connector (J1) on the front panel of Prompt, necessary for the proper configuration of ribbon cable between the user system and Prompt.

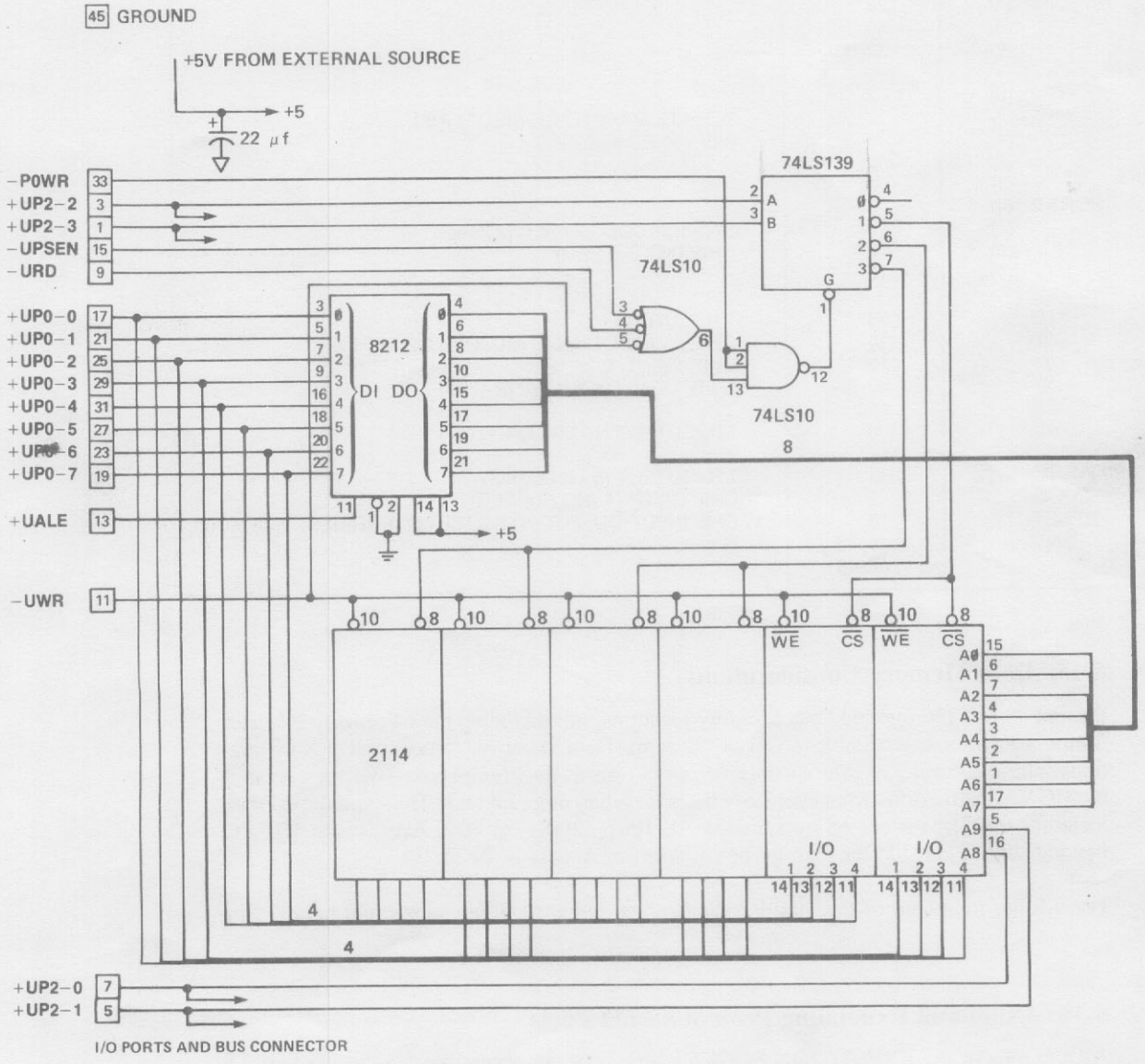


Figure 6-2. Design for "von Neumann" Expansion Memory

Table 6-1. Pin List for I/O Ports and Bus Connector

Signal Name	Pin No.	Buffer Characteristic
BUS (0)	17	
(1)	21	
(2)	25	
(3)	29	
(4)	31	3-STATE BIDIRECTIONAL
(5)	27	
(6)	23	
(7)	19	
PORT 1 (0)	18	
(1)	20	
(2)	22	
(3)	24	8748 PSEUDO BIDIRECTIONAL
(4)	26	CHIP (NO BUFFER)
(5)	28	
(6)	30	
(7)	32	
PORT 2 (0)	7	
(1)	5	3-STATE MAPPED BIDIRECTIONAL with 100 Ω IN SERIES
(2)	3	
(3)	1	
PORT 2 (4)	4	
(5)	6	8748 PSEUDO BIDIRECTIONAL
(6)	8	CHIP (NO BUFFER)
(7)	10	
+ALE	13	TTL OUTPUT (10
+T0	14	CHIP BIDIRECTIONAL (CLOCK), 2.2K Pullup
+T1	12	CHIP INPUT, 2.2K input
−INT	49	1 TTL LOAD (MON. GATED)
−PSEN	15	
−RD	9	TTL OUTPUT (10 LS LOADS)
−WR	11	
−P0 WRITE	33	TTL OUTPUT (5 LS LOADS)
−PROG	2	CHIP OUTPUT (NO BUFFER)
−RESET	16	CHIP INPUT/OUTPUT (SYS RESET OVERRIDES), 2.2K pullup
GND	45, 46 47, 48	Ground

6-15. Data Memory Considerations

Prompt 48 has 256 internal Data Memory locations, not including the 64 on-chip Register Memory locations, accessible to you as "External Data Memory," via the MOVX instructions. These 256 bytes of external data memory — inside the Prompt box — will be accessed by MOVX instructions whenever LSN P2 is less than or equal to 3. That is, external data locations 0 will be accessed by addresses 0, 100_{16}, 200_{16}, or 300. Accesses to 400 and beyond will be outside the Prompt box (except in Access = 2, 5).

For a fuller treatment of the vital P2 subject, see the appropriate subsection below.

6-16. Using and Expanding Prompt 48 I/O Ports

All I/O pins of the EXECUTION SOCKET processor are accessible via the I/O PORTS AND BUS CONNECTOR (see Table 5-1). Some lines are buffered inside the Prompt, and therefore differ somewhat from a standalone MCS-48 device.

The connector pins designated port 1 are not buffered; they are connected directly to the EXECUTION SOCKET computer.

The connector pins designated port 24 through 27 are not buffered; they are connected directly to the EXECUTION SOCKET computer. However, the pins designated port 20 through 23 (the LSN P2) *are* buffered. Ordinarily the P2 MAP function [2] specifies whether the lines of port 2 are to be used as input or output. The map enables appropriate port 2 buffers, and allows you to examine/modify port 2 ("register 47") from the Prompt panel. The default for P2 MAP is that all lines be output. *Important:* Do not confuse the P2 MAP with the port itself (register 47). They are entirely different.

If LSN P2 is to be used as input, you must map it accordingly, and execute from on-chip program memory only (1K or less), ACCESS = 3 or 5. The MSN P2 can be input whenever it is so mapped.

The Prompt does not usually allow any P20-P23 pin to be both input and output. The one exception is using an 8243 I/O expander (and ACCESS = 1, 4). Then Prompt ignores P2 MAP and automatically switches the LSN P2 buffers between input and output, as signalled by the PROG pin.

The connector pins designated BUS 0 through 7 (also known as port 00 through 07) are buffered. In access codes 0 and 3 will latch. These lines will be latched outputs. No inputs are allowed, and memory may not be expanded outside Prompt box. The MCS 48 processor can, however, execute monitor programs or user programs from writable program memory, and these bus transactions do not appear on the latched PORT 0. Only the OUTL P0 instruction or any instruction generating writes (WR) will alter the latched BUS (P0) contents; ANL P0 and ORL P0 instructions have no effect.

If you are using BUS for input, for strobed output or for expansion memory (and memory-mapped I/O) then you will select access 1, 2, 4 or 5. Prompt requires that LSN P2 >3 for access outside the Prompt box, including input, strobed output expansion memory and memory-mapped I/O.

Prompt provides a signal called –P0WR which goes low whenever Prompt's port 0 latch buffers are driving out of the box. You may use this signal to disable any of your user system bus drivers which might be driving into the Prompt box.

6-17. P2 Map, LSN of P2, Access Code Considerations

P2, or Port 2, is one of the MCS-48 processor's three 8-bit parallel I/O ports. It acquires special significance because it is used to output the Most Significant 4 address bits of transactions with both Program Memory and Data Memory (the 8 Least Significant bits of the 12-bit address are provided by the BUS port). Only the Least Significant Nibble of P2 is required for this purpose; thus the numerous references in this manual to LSN of P2 considerations.

The P2 Map is given by the user through a panel command (see Paragraph 5-15) to establish the signal direction on a pin by pin basis within Port 2. (The default condition is "output.") The Most Significant Nibble of P2 may be freely mapped as "input" or "output" according to the user's needs. But because of the Prompt 48 environment, the LSN P2 Map could compete with Prompt's drivers under certain Access Modes. This requires explanation on a mode-by-mode basis:

6-18. Modes 0, 2, or 5: Map LSN as Output. LSN pins are used in these modes by the Monitor to select various internal memories of the Prompt 48 and therefore must not be affected by input devices. Referring to Figure 6-3, we can see the data path is P2, H, J, K. If LSN is mapped input, data path J1, A, D, G, H could foul things up.

6-19. Mode 1 or 4: Mapping is Don't Care. LSN is used by the user to select various external memories, I/O chips, and/or 8243 Port Expander chips he may have connected to J1. Being select lines, the LSN function will always be output except if using an 8243 Port Expander. In Figure 6-3 the path is H, G, B, E, J1. The LSN mapping mechanism is actually bypassed in these modes and is therefore immaterial. If it is mapped as output, the contents are saved by the monitor during debug. If using an 8243, on a MOVD A, Pn command, the path switches to J1, F, B, G, H.

6-20. Mode 3: Mapping May Be Input or Output as the User Requires. In this mode we are running a program less than 1k long which resides on the processor chip. With Input mapping the path is J1, A, D, G and H. With Output mapping the path is P2, H, J, C, A, J1. You might notice that if the Monitor takes control (due either to single-step, with-break, or Monitor interrupt pressed) the last data on the 4-bit latch is held and the P2 Map is temporarily switched to Output. Again, this is to prevent possible input lines from affecting the internal memory select lines.

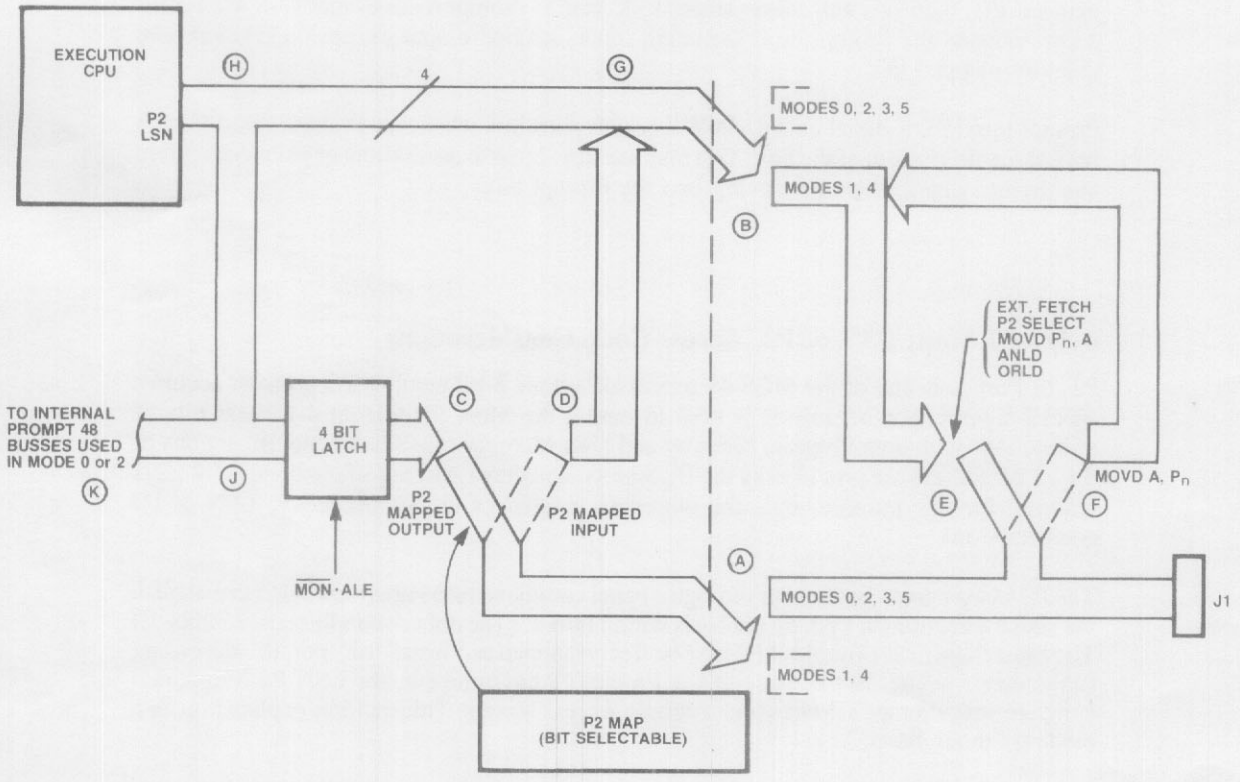


Figure 6-3. PROMPT 48 Port 2 Bus Structure

6-21. LSN P2 Considerations. Prompt 48 is designed to automatically select the correct program memory: addresses 0 to 3FF (1K-1) are inside the box, either on-chip EPROM or its writable substitute. Addresses 400 to FFF (1K or greater) are to expansion memory, which you provide outside the box. The correct memory is automatically selected by using the LSN P2 as an "inside/outside resource switch".

If LSN P2 ≤ 3 ($< 1K$), then all accesses are to resources inside the Prompt box. If LSN P2 > 3 ($\geq 1K$), all accesses are to resources outside the Prompt box.

6-22. Using the Serial I/O Port

Prompt 48 is shipped from the factory with its default options strapped for use with the Prompt-SPP Option, but may alternatively be strapped for 20-mA current loop Teletypewriter terminal or for any RS232c-compatible terminal.

The Serial I/O Interface communicates with an external I/O device via a 26-pin double-sided PC edge connector (J2), 0.1 inch centers. An external device can be connected to J2 using a 3M 3462-0001 flat cable connector or one of the following soldered connectors: TI H312113 or AMP 1-583715-1. Table 6-2 provides a pin list for connector J2.

Expansion program memory is automatically selected by the most significant nibble of PC, which is strobed through LSN P2 during program memory fetch (PSEN/). However, expansion data memory (or memory-mapped I/O) which is outside the Prompt box will be selected only if LSN P2 > 3. That is, if either P22 or P23 = 1.

For example, to access data memory outside the Prompt box (MOVX) you may need to insert in your program LSN P2 > 3. (If LSN P2 ≤ 3, MOVX will access the external data memory inside the Prompt box.)

When your MCS 48 system finally stands alone, without Prompt, the LSN P2 requirement is obviated.

From the Prompt 48 panel you can [EXAMINE/MODIFY] [PROGRAM MEMORY] in the range 400-FFF. Prompt's monitor will generate reads (RD) and writes (WR) to whatever expansion devices — program memory, data memory, or memory-mapped I/O — are addressed by the 12 bits LSN P2 BUS. The [EXAMINE/MODIFY] [DATA/MEMORY] button only accesses the 256 bytes external data memory inside the Prompt box.

Table 6-2. Connector J2 Pin Connections

Pin		Pin	
1	CHASSIS GND	2	+5V (if 31–32 strapped)
3	TRANSMITTED DATA	4	
5	RECEIVED DATA	6	TTY RD CONTROL
7	REQ TO SEND	8	
9	CLEAR TO SEND	10	
11	DATA SET READY	12	
13	GND	14	Tx CLK/DATA TERMINAL RDY
15	DATA CARRIER RETURN	16	TTY RD CONTROL RETURN
17		18	
19		20	
21		22	RECEIVE CLK/TTY Rx
23	TTY Rx RETURN	24	TTY Tx RETURN
25	TTY Tx	26	GND

Table 6-3. Serial I/O Port Strapping Options

Prompt-SPP	TTY	RS232
1–2 (J2–1 = GND)	1–2	1–2
3–4 (J2–6 = RD CNTL)	3–4	4–5
6–7 (J2–14 = DSR)	6–7	6–7
9–12 (RTS = CTS)	9–12	9–12
10–11 (J2–7 = Always CTS High	10–11	10–11
14–15 (TXC = RXC)	14–15	14–15
17–18 (J2–23/32)	17–18	16–17
19–20 (2400 BAUD)	19–20	See Table 6-4
21–27 (2400 BAUD)	21–25	See Table 6-4
31–32 (J2–2 = +5V)	Disconnect 31–32	Disconnect 31–32
All others	Disconnected	
CAUTION: Unrelated to the serial interface may be a jumper from 29–30. This must remain untouched at all times.		

Table 6-4. Baud-Rate Selection

Baud Rate	Strapping Connections
4800	21–26 (19 & 20 DON'T MATTER)
2400	21–27 (19 & 20 DON'T MATTER)
1200	21–28 (19 & 20 DON'T MATTER)
600	21–22 (DISCONNECT 19 & 20)
300	21–23 (DISCONNECT 19 & 20)
150	21–24 (DISCONNECT 19 & 20)
75	21–25 (DISCONNECT 19 & 20)
110 (TELETYPE)	21–25 AND 19–20

6-23. Interfacing to a Teletypewriter

The teletypewriter must receive the following internal modifications and external connections, for use with the Prompt 48.

6-24. Internal Modifications. Complete the following internal modifications.

a. The current source resistor value must be changed to 1450Ω. This is accomplished by moving a single wire (see Figure 6-8).

b. A full duplex hook-up must be created internally. This is accomplished by moving two wires on a terminal strip (see Figures 6-6 and 6-10).

c. The receiver current level must be changed from 60 mA to 20 mA. This is accomplished by moving a single wire (see Figures 6-7 and 6-10).

d. A relay circuit must be introduced into the paper tape reader drive circuit. The circuit consists of a relay, resistor, a diode, a thyractor and a suitable mounting fixture. This change requires the assembly of a small "vector" board with the relay circuit holes in the base plate (see Figure 6-4). The relay circuit may then be added without alteration of the existing circuit (see Figures 6-4 and 6-6). That is, wire "A" (Figure 6-10), to be connected to the brown wire near its connector plug. The "line" and "local" wires must then be connected to the mode switch (see Figures 6-6 and 6-10).

6-25. External Connections. Complete the following external modifications.

a. A two-wire receive loop must be created. This is accomplished by the connection of two wires between the teletypewriter and the Prompt 48 in accordance with Figure 6-10.

b. A two-wire send loop similar to the receive loop must be created.

c. A two-wire tape reader loop connecting the reader control relay to the Prompt 48 must be created.

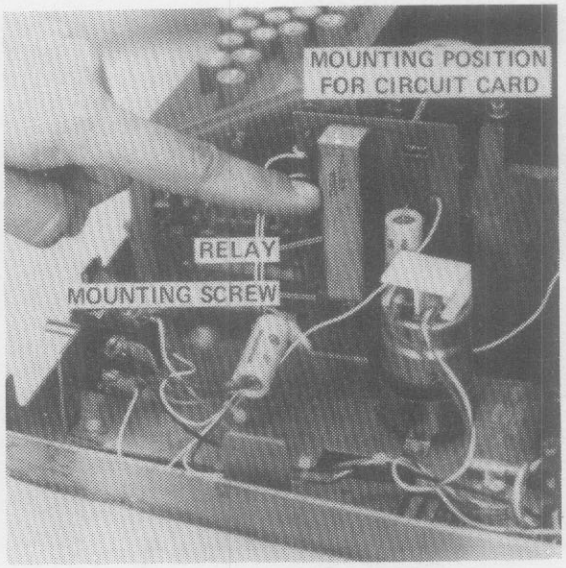


Figure 6-4. Relay Circuit (Alternate)

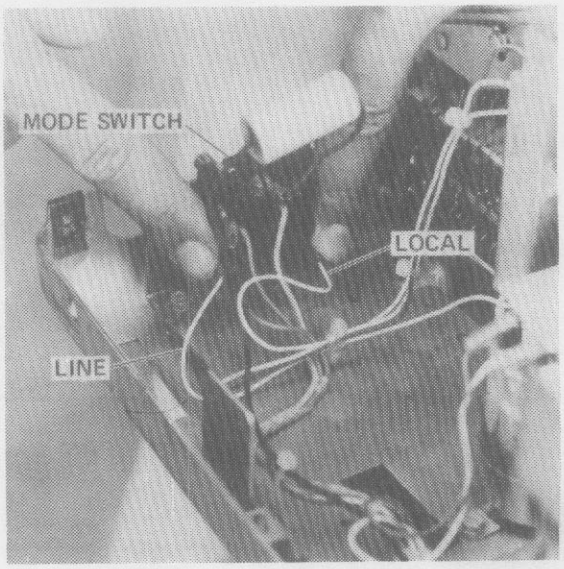


Figure 6-6. Mode Switch

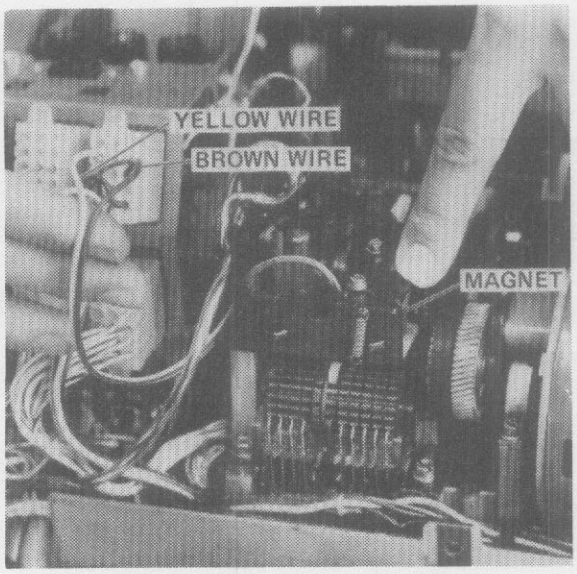


Figure 6-5. Distributor Trip Magnet

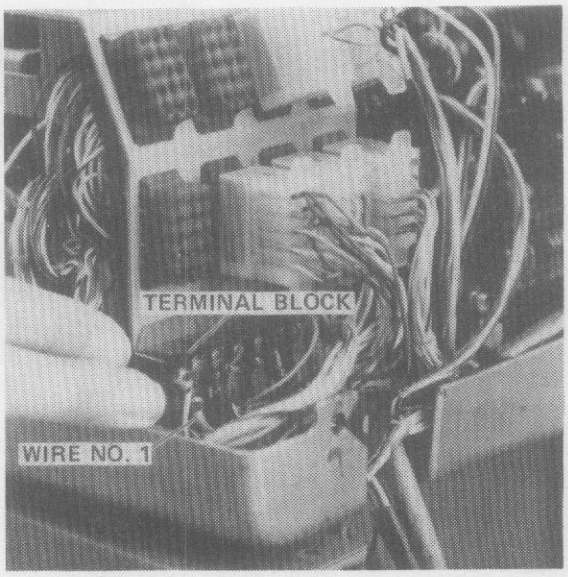


Figure 6-7. Terminal Block

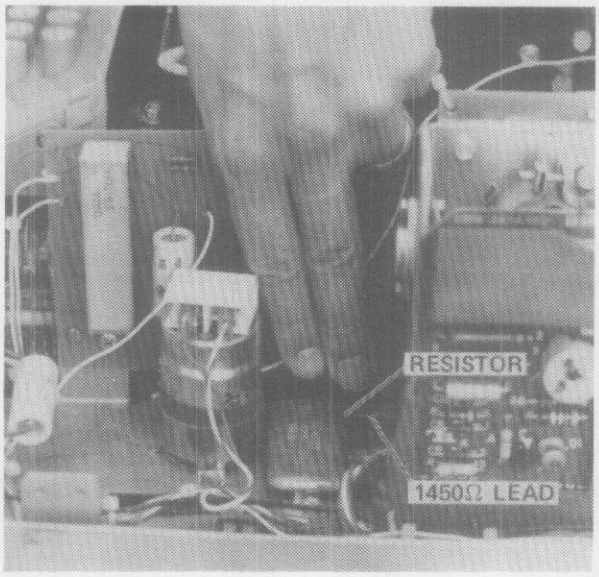


Figure 6-8. Current Source Resistor

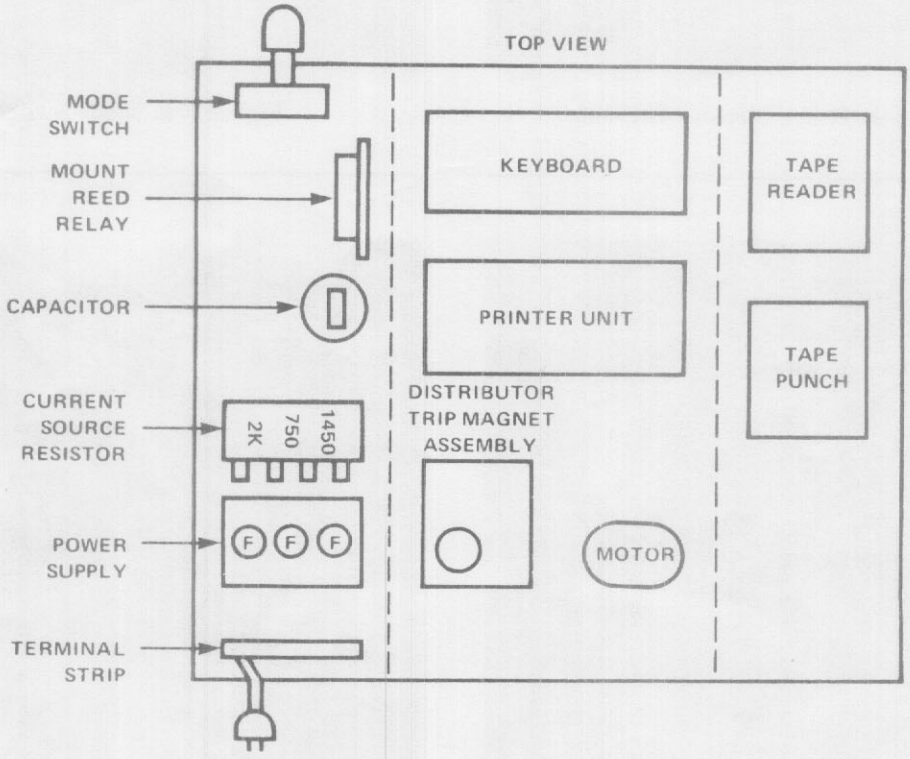


Figure 6-9. Teletypewriter Layout

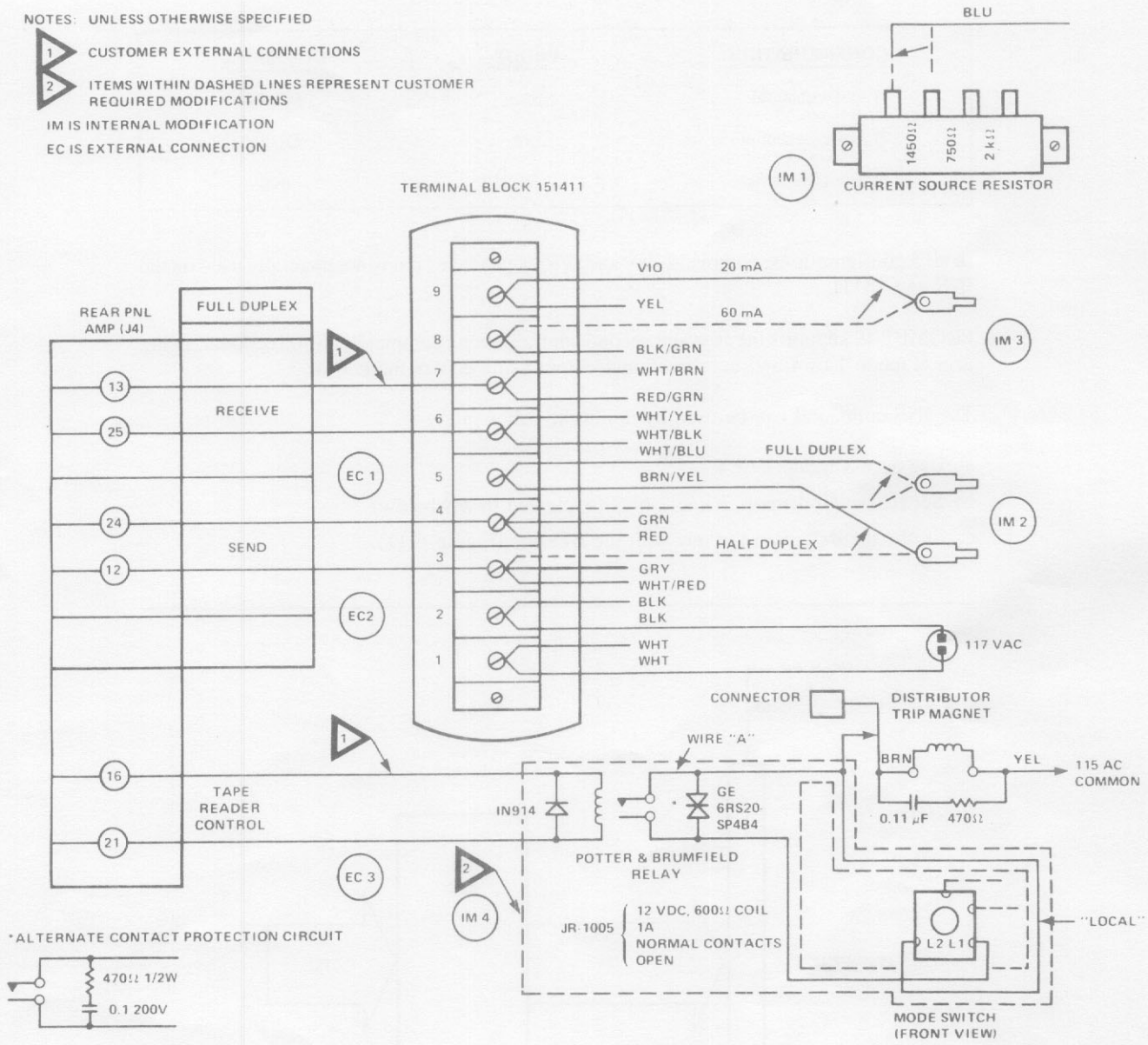


Figure 6-10. Prompt/TTY Wiring Diagram

6-26. Questions Most Often Asked

6-27. Use of INS A, BUS.

At the chip level, the MCS-48 BUS port was designed to work in one of the following configurations, not in a combination of these modes.

CONFIGURATION	IN/OUT	COMMAND
1) Bi-Directional	both	MOVX
2) Uni-Directional	out	OUTL
3) Uni-Directional	in	INS

In all 3 configurations, command RD/ and WR/ is produced but is not generally used on the INS and OUTL.

PROMPT 48 supports the first and second configurations completely: bi-directional, using access mode 1 or 4 and uni-directional output using access mode 0 or 3.

The INS command can be used by doing the following:

a. Use access mode 1 or 4

b. Set (Drive High) port 2 line 2 or 3 (explained in #8 below)

c. Strobe the data onto the bus with the RD line (Figure 6-11).

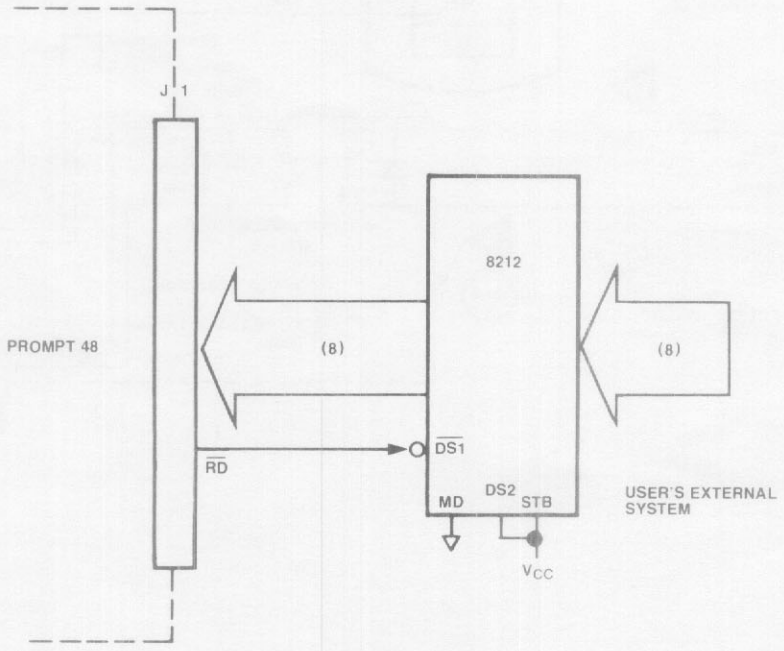


Figure 6-11. Strobed Data Input

Figure 6-12 shows the reason for the above steps. Access mode 1 or 4 enables a bi-directional driver and tri-states a latch that holds the data on an OUTL Bus,A command. Setting P22 or P23 deselects internal PROMPT 48 memories. Data must be strobed onto the bus or else the inputs would fight the 8216 drivers which are driving out when RD is inactive.

6-28. RAM And I/O Selection

On MCS-48 systems, the MOVX command is used for data and I/O transfers with R0 or R1 as a pointer. The addressing capability is then limited to 1 page (256). This is expanded to 4K by using P20-23, decoded to 16 page selects. Internally the PROMPT 48 requires the first 1K addresses, i.e., P22 and 23 low. There are 2 consequences of this:

a. To access the 256 byte user RAM that's inside the PROMPT 48, the user program must output 0's to P22 and 23. (drive low). P20 and P21 are 'don't care'.

b. To select data and I/O that has been bussed to J1, either P22 or P23 has to be driven high (logic 1). This deselects any internal memory.

Summarizing the above:

	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	P23	P22	P21	P20	←			(RO) (R1)				→
Internal Prompt 48 →	0	0	X	X								
External Selection (Mode 1 or 4)	0	1	X	X								
	1	0	X	X								
	1	1	X	X								

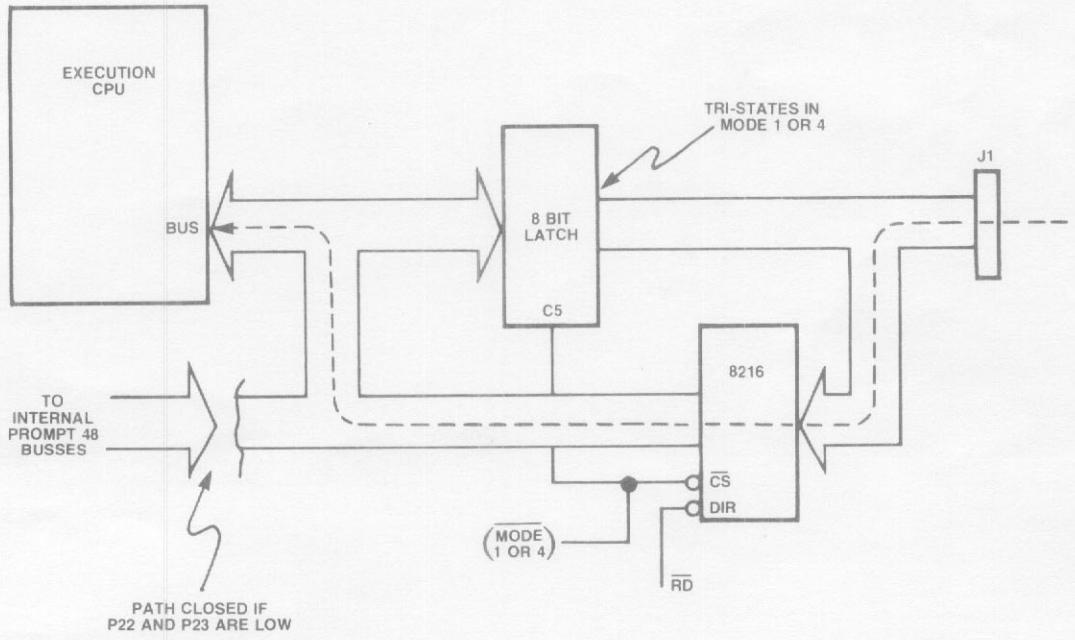


Figure 6-12. Data Path Within PROMPT 48 Using INS A, BUS

6-29. TTY and CRT Peripherals Are Used Only For Dumping Or Reading Paper Tape

The keyboard input is not a substitute for the keypad on the PROMPT 48.

6-30. Speed Degradation Occurs When "GO WITH BREAKPOINTS".

This is due basically because the operation is a replica of single-stepping. This means that after every instruction the monitor re-takes control, saves the processor state, checks the PC against the eight breakpoints, then restores the processor state and goes back to the User mode. — If your program has timing loops in it, the speed of execution will be substantially lengthened.

6-31. When Using PROMPT 48 System Calls, Do Not "GO WITH SGL. STEP" Or "GO WITH BREAKPOINT"

The monitor is like a lot of us; it does not handle self-examination very well!

Do not try to read the program from an 8048 through the PROMPT Programming Socket. It is meant for EPROMs only!

APPENDIX A
A FAMILIARIZATION EXERCISE

Voltage Selection

Check the voltage selection switch visible on the PROMPT rear chassis. Ensure it is set to your local mains (line) voltage; if not, open the PROMPT box and remove the switch, locking plate, set to proper voltage, and reinstall. If you change the switch setting you probably should change the fuse. Plug the unit in and turn power on (switch is on rear chassis).

Handling The Processor

Carefully remove either processor (8748 or 8035) from the conductive foam. The processors are mechanically and electrically fragile, and will shear through the chip and package if dropped. Do not apply uneven pressure to the processor—for example, pushing on the lid or even on both ends of the package can destroy your processors.

Insert In Execution Socket

Pull EXECUTION SOCKET locking arm up towards you. Insert either processor in EXECUTION SOCKET, and lock in place by pushing locking arm flush with panel.

Reset The System

Press [SYS RST] to reset system. ACCESS = 0 should appear on display. If not, try repeating the above steps with the other processor, and notify your Intel service center or representative of the problem.

INTEL SERVICE HOT LINES:

From locations within California call toll free—
(800) 672-3507

From all other U.S. locations call toll free—
(800) 538-8014
TWX: 910-338-0026
TELEX: 34-6372

From Alaska, Canada or Hawaii call—
(408) 987-6218

From Europe call—
(322) 72-3565
TELEX: 846-24695

There are six access codes, numbered 0 to 5. Whenever you power up or reset the PROMPT system, ACCESS will be set to 0. We will explain how to change ACCESS codes and why they are useful momentarily.

Your PROMPT 48 system is fundamentally different from all other computers: This is the first time the processor has been outside the box. You can safely remove the processor(s) from the panel sockets at any time, provided a processor is properly inserted in the EXECUTION SOCKET whenever you insert or remove a processor from the PROGRAMMING SOCKET.

Always insert the EXECUTION SOCKET processor first, and remove the PROGRAMMING SOCKET processor first.

The EXECUTION SOCKET processor is always executing either the monitor or your (user) programs. When ACCESS = 0 or the prompting "-" hyphen character appears, then the monitor is ready to accept COMMANDS or FUNCTIONS.

Let us first exercise the monitor COMMANDS. Notice that the command buttons have been color-coded white and blue. Throughout this exercise each bracket pair [ ] is a button to be pressed.

Mnemonics enclosed in braces, e.g., {SMA} or {DIR} are parameters, usually self-explanatory, such as SMA Starting Memory Address, or DIR, Direction. You have to push two or three hex buttons for each parameter.

Examining and Modifying Registers

Press [EXAMINE/MODIFY] [REGISTER].

Now you may enter any number (address) of any register you wish to examine and/or modify.

Press [0] for register 0.

Now press NEXT [,]—the comma button—to "open up" register location 0. The contents (random) are displayed.

Now notice you can "roll in" any data that you want in that open register. Press [0]. Suppose you want 1. Press [1]. Suppose you want 22. Press [2] [2].

The monitor allows you to roll data into any location as long as it is open. A location is open until closed by terminating the command (press [.]EXECUTE/END) or by opening some other location.

There is an easy way to open and close locations in succession.

A prompting hyphen character "-" should now be displayed. If not, press [.]EXECUTE/END. Open register location 0 again—press [EXAMINE/MODIFY] [REGISTER] NEXT [,]. The data you left in register 0 (22?) should appear.

At this point you have opened register 0. To open register 1 (and close 0) simply press NEXT [,]. To open 2 and close 1 press NEXT [,] once again. To go backwards, opening previous locations, press [PREVIOUS]. Press [PREVIOUS] again. Register 0 should be open now. Close it by terminating the command [.]EXECUTE/END.

The MCS-48 has 64 registers, numbered 0 to 3F hexadecimal. All PROMPT 48 addresses and data are entered and displayed in hexadecimal. There are some special purpose locations, such as the accumulator, which we have assigned register numbers:

Number	Location	Format
40	ACCUMULATOR	
41	TIMER	
42	PSW	CY | AC | F0 | BS | F1 | S_2 | S_1 | S_0
43	PCL	
44	PCH	
45	PORT 0 (BUS)	READ-ONLY
46	PORT 1	READ-ONLY
47	PORT 2	
48	MISC	Counter Run | Timer Run | Timer Flag | Nested Fr Int | Will En Int | Mem Bank | T1 | T0

Note that the PSW (register 42) as EXAMINE/MODIFied from the Prompt panel includes the Flag 1 F1 test bit. It's been added for ease of debugging. The real MCS-48 PSW as accessed by your program does not contain F1.

Note that ports 0 and 1 (registers 45, 46) cannot be modified by EXAMINE/MODIFY. These can only be read.

The bits of MISC (register 48) require explanation:

COUNTER RUN—if your program uses the MCS-48 timer/event counter as an event counter you must manually set this bit to "1". Otherwise PROMPT assumes you will use the timer/event counter as a timer. Your program should still use STRT CNT, STRT T, and STOP TCNT instructions as usual. The COUNTER RUN bit is the only way the PROMPT monitor can tell whether you are using the EVENT COUNTER instead of the TIMER. It allows the monitor to properly suspend and restart the timer/event counter when a "break" occurs.

The transition from user program to monitor program is called a "break." During breaks the PROMPT monitor takes pains to save the state of the broken user program so that it can be restored if you resume execution.

For example, TIMER RUN will be set 1 on break if the timer is running. If you clear this bit to 0 the timer will not be restarted when execution is resumed. You should not need to change this bit.

TIMER FLAG allows you manually to examine and modify the timer flag.

NESTED FROM INTERRUPT will be set to 1 if you have broken during a routine servicing a monitor interrupt. This bit is used for monitor housekeeping, and ordinarily should not be changed.

WILL ENABLE INTERRUPT—The monitor sets this bit to 1 if it will enable interrupts when you resume execution. You should not need to change this bit.

MEM BANK is the memory bank select bit, the high order bit address bit for fetches from program memory.

T1 and T0 are the test inputs (READ ONLY).

Correcting Errors (Clear Entry)

If you ever enter wrong COMMANDS, HEX DATA or FUNCTIONS you can easily correct it. Of course, if a location is "open" (as in EXAMINE/MODIFY) you merely roll in data until you are satisfied it is correct. At these times—when a location is "open"—the PREVIOUS button will open the previous location.

But notice the PREVIOUS button is also labeled CLEAR ENTRY. At all other times, whenever a location is not "open", pressing CLEAR ENTRY will abort a command or clear an error. Thus the CLEAR ENTRY/PREVIOUS button does double duty, and it does what makes sense.

For example, press

[EXAMINE/MODIFY] [CLEAR ENTRY].

Press [EXAMINE/MODIFY] [EXAMINE/MODIFY] [CLEAR ENTRY].

Press [EXAMINE/MODIFY] [REGISTER] [CLEAR ENTRY].

Whenever the monitor detects an error, such as Ud (undefined function) it will spell "Err" and is ready to accept new commands with the next keystroke.

Examining And Modifying Program Memory

Besides the 64 registers there are 1K bytes of EPROM program memory on the 8748 chip. This program memory is erasable, programmable read-only memory. It is non-volatile, and can be programmed in seconds, but it requires several minutes to erase.

To speed your design efforts, 1 Kbyte of RAM (read-write) program memory has been provided on the PROMPT system. This can be used in place of the on-chip EPROM program memory. It is volatile, but can be quickly and conveniently examined and modified.

For example, press

[EXAMINE/MODIFY] [PROGRAM MEMORY] starting at location
[0] NEXT [,].

Program memory location 0 is now "open" and any instruction can be rolled in. The code for increment accumulator (INC A) is 17. Enter it. Press

[1] [7] NEXT [,].

Now enter the instruction "jump to 0", whose codes are 04, 00.

Press [0] [4] NEXT[,] [0] [.]END.

You have entered a simple program. To verify it, again open up program memory location 0 and step through the next locations.

Press [EXAMINE/MODIFY] [PROGRAM MEMORY] [0]

NEXT[,]
NEXT[,]
NEXT[,].

Note you can step backwards, as with registers. Press

[PREVIOUS]
[PREVIOUS]
[PREVIOUS]

and then [.]END the command.

We will run the simple program momentarily.

Examine/Modify Data Memory

The 64 registers on each MCS-48 chip are the primary "register memory" for data. But should more data memory be required your MCS-48 system may be expanded with "external" data memory.

The PROMPT system provides 256 such external data memory locations number 0–FF. You can examine and modify them by pressing

[EXAMINE/MODIFY] [DATA MEMORY]

[0] NEXT[,]

which opens location 0. You can roll in data and step through the next or previous locations as with the other EXAMINE/MODIFY commands.

MCS-48 manuals refer to such data memory as "external" because it is outside the chip computer. But 256 bytes of this memory are inside the PROMPT box. Thus we will refer to the external data memory inside the PROMPT box.

You can add more data memory than the 256 bytes provided in PROMPT. Simply interface expansion memory to the I/O ports and BUS CONNECTOR, at address 1K (400_{16}) or greater. Then this expansion data memory is examined and modified by the [EXAMINE/MODIFY] [PROGRAM MEMORY] keys, and appropriate addresses.

Access Codes [A]

Now we can explain the ACCESS codes, and run the program just entered in writable (RAM) program memory.

ACCESS codes allow you to specify

a. which program memory you will use, either WRITABLE (RAM) in the PROMPT box or READ ONLY (ROM/EPROM) on the CPU chip

b. how you will use Port 0 (BUS). It can be used either
   1. as a *port*, latched on output. Under this access OUTL PORT 0 would work;
   2. as a *bus*, to address *expansion* memory and I/O outside the PROMPT box; or
   3. as a *bus*, to address the PROMPT *system* monitor *memory* and *I/O* devices. This mode would be used if your user program wanted to talk directly to the PROMPT keyboard, displays, or serial channel. A listing of the system monitor routines and their use is in Appendix B.

The first two uses of Port 0 (as latched port or outside *expansion* bus) will be more common. Of course, programs can be run from READ ONLY (on-chip) memory or from its WRITABLE (RAM) replacement.

The access codes are summarized:

Access Code	Program Memory	System I/O and System Calls	Expansion Memory and I/O	OUTL Port 0
0	WRITABLE (RAM)	no	no	yes
1	WRITABLE (RAM)	no	yes	no
2	WRITABLE (RAM)	yes	no	no
3	READ ONLY (ON CHIP)	no	no	yes
4	READ ONLY (ON CHIP)	no	yes	no
5	READ ONLY (ON CHIP)	yes	no	no

You can change access codes (or enter any other system commands or functions) whenever the power-up message "ACCESS=0" or prompting hyphen "-" appears.

Here's how. Press [A] [1] [.] END. You have selected ACCESS code 1.

Press [A] [0] [.] END to return to ACCESS code 0.

P2 Map [2]

Just as ACCESS CODES establish how Port 0 (BUS) will be used, the Port 2 MAP command establishes the DIR (direction) of each Port 2 line. The bits of DIR map each line of Port 2: IN=1, OUT=0.

[2] {DIR} [.]

On power-up and [SYS RST] the monitor assumes all lines should be output, and therefore clears the P2 MAP to zero.

Recall that MCS-48 processors use the least significant nibble (LSN) of Port 2 to address off-chip (expansion) program memory and I/O ports. Thus the LSN P2 MAP, the contents of LSN P2 and the ACCESS code are related.

If you have selected expansion memory and I/O (ACCESS = 1 or 4) then the MAP for LSN P2 is ignored because LSN P2 must be bidirectional to work with the 8243 I/O expander. PROMPT detects when signals must flow in or out through LSN P2, and switches buffer drivers accordingly.

At any other time that you access off-chip resources—whether writable program memory, external data memory, or expansion I/O—the LSN P2 should be mapped output. Thus if ACCESS = 0 or 2, the P2 MAP should be X0, where X is user-defined.

Said another way, LSN P2 can be used as input and mapped input only if ACCESS = 3 or 5 and certain cautions about its contents are observed. We recommend that you use and map LSN P2 as input only if P0 (bus) is always output, that is if your program is less than 1K bytes and on-chip, in EPROM. PROMPT monitor calls, PROMPT system I/O, and accesses to data memory should not be done.

Think of LSN P2 as an inside/outside resource switch. If the LSN of Port 2 is ≤ 3, corresponding to $\leq 1023_{10}$, then all memory accesses are *inside* the PROMPT box, to

a. The on-chip program memory, or

b. its writable program memory replacement, or

c. the 256 bytes "external" data memory ("inside" the box).

If the LSN of Port 2 is greater than 3, corresponding to $>1023_{10}$, the accesses are to

a. Port 0 as an input/output port (ACCESS = 0, 3), or

b. program memory, data memory, or I/O devices *outside* the PROMPT box (ACCESS = 1, 4), or

c. system monitor program memory and memory-mapped system I/O devices inside the PROMPT box (e.g., PROMPT serial channel) (ACCESS = 2, 5).

There are some subtle implications. For a program to access the external data memory inside PROMPT, ensure LSN P2 $\leq$ 3. To input on Port 0, ensure LSN P2 >3.

Remember, LSN Port 2 can be set several ways, by

a. manually [EXAMINE/MODIFY]-ing [REGISTER] 47 (Port 2);

b. executing an OUTL P2, ORL P2 or ANL P2 instruction;

c. allowing the program counter to exceed 3FF (1023_{10}). When PC $\geq$ 400 then program fetches are off-chip. The processor strobes the most significant PC nibble (e.g., 4) through least significant P2 nibble.

Executing Programs (Go No Break)

There are three ways to run a program. See the white-color-coded COMMANDS:

[GO] [NO BREAK]
[GO] [WITH BREAK]
and [GO] [SINGLE STEP].

Let's run the simple program we entered in writable program memory.

```
000 INC A
001 JMP 0
```

First examine the accumulator. Press

[EXAMINE/MODIFY] [REGISTER] [4] [0] NEXT [,]

and remember its contents. Close the accumulator.

[.] END

Now enter [GO] [NO BREAK] [0] [.] EXECUTE.

The user program is running in real time, mindlessly incrementing the accumulator. Stop it. Press [MON INT] to interrupt and break to the monitor.

Whenever a break occurs, the program counter address is displayed together with accumulator data.

You can always press [GO] [.] EXECUTE to resume execution at the current program counter address. [MON INT] will break again to the monitor.

Single Stepping Programs (Go Single Step)

Instead of running in real time, you can single-step a program. This is running as though there were a break after every instruction.

Press [GO] [SINGLE STEP] [0] to prepare for single-stepping at location 0. Each time you press

NEXT [,]
NEXT [,]
NEXT [,] (etc)

one instruction is executed and a break occurs. Press [.] END.

As with the GO NO BREAK command, you may omit the starting address (0) and resume single-stepping from the current program counter address. For example, press

[GO] [SINGLE STEP] (no start address needed)

NEXT [,]
NEXT [,]
NEXT [,] (etc.)
[.] END.

Setting Breakpoints (The [B] Function)

When you are debugging larger programs you will want selectively to set several breakpoints. PROMPT allows you to set as many as eight breakpoint addresses.

Press the [B] function. Now open up breakpoint 0. Press:

[0] NEXT [,].

Probably it will contain random numbers. You can step through the entire breakpoint table, opening NEXT or PREVIOUS table entries by pressing

NEXT [,]
NEXT [,]
NEXT [,]
[PREVIOUS]
[PREVIOUS] and so on.

Press [.] END to terminate the command.

To clear all breakpoints, press

[B] [.] END.

Now examine the breakpoint table. Press

[B] [0]

NEXT [,]
NEXT [,]
NEXT [,]
and so on
[.] END

Let us set a breakpoint at each instruction in our simple program. Suppose breakpoint 2 is set at location 0, and breakpoint 3 is set at location 1. (Breakpoints 0, 1, 4–7 remain unused.)

Enter

```
[B] [2] NEXT [,] [0]
        NEXT [,] [1]
                 [.] END.
```

Check the breakpoint table. Enter

```
[B] [0] NEXT [,]
        NEXT [,]
        NEXT [,]
        NEXT [,]
        NEXT [,]
        NEXT [,]
                 [.] END.
```

Running With Breakpoints (Go With Break)

```
Now press [GO]  [WITH BREAK] [0]
          NEXT [,]
          NEXT [,]
          NEXT [,]
          NEXT [,]
          NEXT [,]  and so on.
```

After each user instruction the monitor is run; the user program counter is compared with entries in the breakpoint table. If the user PC is not at breakpoint, execution is resumed.

Of course this breakpoint checking after each user instruction requires many monitor instructions. GO WITH BREAK runs programs about 2,000 times slower than real time.

You can selectively clear breakpoints. Pressing

```
[B] [3] [.] END
```

will clear breakpoint 3. Try

```
[GO] [WITH BREAK] NEXT [,]
                  NEXT [,]
                  NEXT [,]
                  NEXT [,]
                       [.] END
```

As with the other GO commands, the starting address is optional. If you omit it, execution begins at the current program counter.

You are now familiar with all of PROMPT 48's commands, and a number of its functions.

Let us cover the remaining functions.

Clear Memory [C]

Allows you to clear either register, program, or data memory. Specify starting and ending memory address.

For example:

[C] [REGISTER] [0] NEXT [,] [3] [F] [.] END

clears all 64 registers.

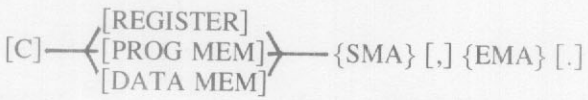
[C] [PROGRAM MEMORY] [0] NEXT [,] [3] [F] [F] [.] END

clears 1024 program memory locations.

[C] [DATA MEMORY] [0] NEXT [,] [1] [.] END

clears external data memory locations 0 and 1.

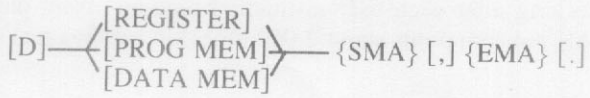
We compactly describe this function as



where SMA is starting memory address, EMA is ending memory address.

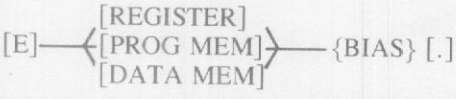
Dump From Memory

Dumps register, program or data memory to paper tape in the standard Intel HEX FORMAT. Assumes a teletypewriter has been interfaced to the PROMPT 48 via a PROMPT-SER cable. See details in Appendix C. With this function you can prepare a paper tape specifying your program memory pattern for volume ROM (8048) orders.



Enter (Read) Into Memory [E]

Enters into register, program or data memory the contents of a paper tape punched in the standard Intel HEX FORMAT.

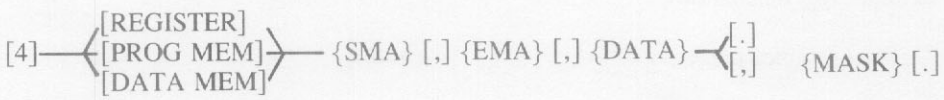


The HEX FORMAT includes both data and load addresses. A bias (ordinarily 0) is added to the load addresses allowing you to offset where anything is entered.

Notice a little "r" appears when you press E. This stands for read. We have already used E to stand for examine/modify.

Byte Search Memory [4]

Searches REGISTER, PROGRAM or DATA memory for one byte of data with optional mask.



For example, press [4].

"S1" appears, indicating a search for one byte. Now press [PROGRAM MEMORY]. Notice "S1" becomes "SP".

Let us search between program locations 0 and 3FF for the data pattern 0. Enter

[0] [,] [3] [F] [F] [,] [0] [.]

The function should find the first zero at location 2. Other occurrences of zero may be found by repeatedly pressing

NEXT [,]
NEXT [,]
NEXT [,]
.
.
.

until the ending memory address is passed or [.] END is pressed.

Think of the mask as a pattern of ones and zeroes. The ones select the bits of each byte which will be compared; the zero-masked bit positions don't count.

Formally, search stops if a match is found, that is, for all bits

[DATA] V [MEM CONTENTS] is 0

If an optional mask is entered then only on the bit masked "1" will the exclusive OR test be applied.

Word Search Memory [5]

Searches REGISTER, PROGRAM or DATA memory for two bytes of data with optional two-byte mask

This function works like the one-byte search just described. HDATA is the high byte of data, LDATA is the low byte of data. HMASK is the high byte of mask, and LMASK is the low byte of mask.

Hex Calculator [6]

[6] {DATA} [,] {DATA} [.]

This function simplifies hexadecimal arithmetic by providing you with a built-in three-digit hexadecimal calculator.

For example, press

[6] [0] [,] [1] [.].

Both the hex sum 0+1=1 and difference 0−1=FFF are displayed.

Press

[6] [B] [,] [A] [.]

Both the hex sum B+A=15 and difference B−A=1 are displayed.

Move Memory [9]

[9] — {[REGISTER] / [PROG MEM] / [DATA MEM]} — {SMA} [,] {EMA} [,] {NMA} [.]

This function moves blocks of register, program or data memory (with starting address SMA, ending address EMA) to some new register/memory address NMA.

Finally, there are four PROMPT 48 functions that deal with the EPROM and PROGRAMMING SOCKET.

Fetch EPROM [F]

[F] {SMA} [,] {EMA} [,] {SEP} [.]

The FETCH EPROM function will fetch the contents of on-chip program memory from the programming socket processor into a block of writable (RAM) program memory in the PROMPT box.

The block of writable memory has starting and ending memory addresses SMA and EMA; the starting EPROM address is SEP.

For example,

[F] [0] [,] [3] [F] [F] [,] [0] [.]

fetches the entire EPROM contents into writable program memory.

This function will signal an error if the programming socket processor is not properly seated.

Compare EPROM [8]

The COMPARE EPROM function compares the contents of the on-chip program memory of the programming socket processor with the contents of the writable (RAM) program memory in the PROMPT box.

[8] {SMA} [,] {EMA} [,] {SEP} [.].

If the programming socket processor is properly seated this command will compare each writable (RAM) program memory location in the range SMA to EMA with the corresponding on-chip EPROM program memory in the range starting at SEP.

The hyphen "-" prompting character appears if there are no errors, otherwise the first mismatched EPROM address and data are displayed.

Successive mismatches may be displayed by pressing NEXT [,].

Program EPROM For Debug [7]

This is the most commonly used function for MCS-48 EPROM programming. The PROM to be programmed must be properly seated. The function first ensures that the top sixteen bytes of on-chip program memory have been programmed with special monitor re-entry instructions. These instructions are required to permit debugging, that is to allow transitions from user to monitor programs, and back.

[7] {SMA} [,] {EMA} [,] {SEP} [.].

Then it will program from a block of writable program memory (SMA to EMA) into the EPROM (at SEP).

Each location is verified after programming, and any errors are displayed.

Program EPROM [3]

This function is similar to the function [7] PROGRAM EPROM FOR DEBUG just described.

However, it does not ensure that the top sixteen bytes of program memory contain the special monitor re-entry code. Unless an MCS-48 processor is programmed with these instructions it cannot be debugged using the PROMPT monitor.

Interrupts

The user interrupt [USR INT] key traps the processor to location 3 if interrupt is enabled. The [USR INT] key is ignored whenever in the monitor, e.g., during breaks as in [GO] [SINGLE STEP] or [GO] [WITH BREAK].

The timer/counter interrupt, however, will properly function only in [GO] [NO BREAK]. The processor traps to location 7.

Immediately upon monitor entry (and just before exit) the [USR INT] key can be locked out (or unlocked) via hardware. But the timer/counter interrupt cannot be instantaneously turned on and off this way. Disabling the timer/counter interrupt (DIS TCNTI) takes finite time. Timer/counter interrupts are random with respect to your program, and could easily occur within the monitor before they have been disabled.

Consider jumping on timer flag instead of trapping via interrupt in the early stages of your program development. The timer/counter operates as though it were "ON" only during user program execution—not during breaks (the monitor program).

APPENDIX B
PROMPT 48 SYSTEM CALLS

There are 9 system calls in the PROMPT-48 monitor that allow the user to access the 24 keys and 8-digit display of the PROMPT-48. Below are listed the entry points (for reference purposes) and a brief explanation of each call.

Routine Name	Function	Address (Hex)
KDBIN	Keyboard Status Loop and Data	7EA
KBST	Keyboard Status	7E4
KBIN	Keyboard Data	7E7
REFS	Refresh Display	7E2
BLK	Blank Display	7DC
ENREF	Hardware Enable of Interrupt Refreshing	7DF
DGSTG	Display Hex Digits	7F3
DGOUT	Update Display Buffer	7ED
HXOUT	Decode and Update Display Digits	7F0

NOTE: All calls (except REFS) select MB1, which may necessitate programming SEL MB0 after the call. Access codes 2 or 5 must be selected to use these calls.

KDBIN (Address 7EA)

Function: Reads keyboard. If key is pressed when routine is entered, looping occurs until the key is released and a new key is pressed. Then the character is returned in A and sets F0 if it is not a hex digit, i.e., one of the 8 control keys. Hex keys return exact hex value. Key debouncing is done on this call.

Control Key Values

PREV	= 10H	EXAM	= 14H
PROG	= 11H	GO	= 15H
DATA	= 12H	NEXT	= 16H
REG	= 13H	EXECUTE	= 17H

Reg. used: A, R0, R6, R7, P2, F0

Reg. modified: A, R0, R6, R7, P2, F0

Parameters expected: None.

KBST (Address 7E4)

Function: Checks keyboard status. Returns

C = 0 = no key pressed
C = 1 = key pressed.

Reg. used: A, R0, P2

Reg. modified: A, R0, P2

Parameters expected: None.

NOTE

This also applies to control keys such as "GO" which means your program may catch the "GO" key still depressed from the initiation of the program. There is no key debouncing done on this call.

KBIN (Address 7E7)

Function: Same as KDBIN except this routine reads the keyboard directly and does no status checking. Used with KBST above. Key debouncing is done on this call.

REFS (Address 7E2)

Function: Refreshes one 7-segment character every time it is entered and sequences through the entire display every 8 times. Displays decimal points from decimal point mask (see below). This routine is generally *interrupt-driven* from loc. 3., i.e.:

```
ORG 3
JMP REFS
```

Reg. used: A, R24-R26, R30, R31, P2

Reg. modified: R24-R26, R30, R31, P2

Parameters expected: See figure B1 below.

A. loc. 38-3F: Display buffer. Contains digits to be displayed. (LED bit pattern form. See DGSTG.)

B. loc. 37: Refresh count, i.e., which digit is to be refreshed. Updated every time routine is called so initial value can be 1-8.

C. loc. 36: Decimal point mask. "1" bit in any or all bit positions causes the decimal point to be displayed in the corresponding display position.

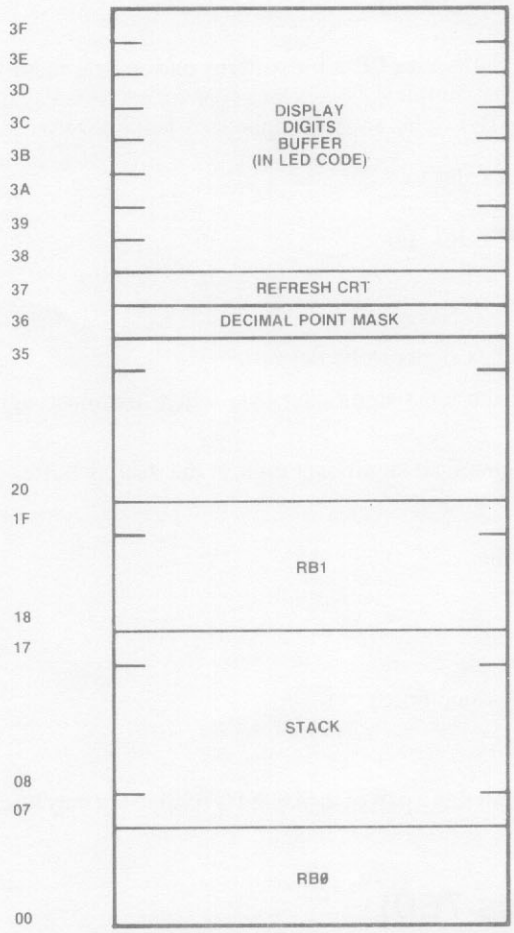


Figure B-1: Register Memory Map

BLK (Address 7DC)

Function: Blanks entire display.

Reg. used: A, R0, R7

Reg. modified: A, R0, R7

Parameters expected: None.

NOTE: Modifies loc. 37-3F.

ENREF (Address 7DF)

Function: Enables the automatic display refresh mechanism on the PROMPT-48 (ORs refresh timer into external interrupt system).

Reg. used: A, R0, P2

Reg. modified: A, R0, P2

Parameters expected: None.

DGSTG (Address 7F3)

Function: Converts hex digits into LED bit patterns and inserts them in the appropriate positions of a display buffer. Buffer is 8 bytes long (one for each display digit) and is located at 38H-3FH. See Figure B-1. This routine suppresses leading zeros.

Reg. used: A, R1, R2, R7, F0

Reg. modified: A, R1, R2, R7, F0

Parameters expected:

A. R7: Number of hex digits to be converted
B. R1: Address of the most significant byte where the most significant hex digit is stored
C. R0: Address of the most significant digit in the display buffer that is to be updated (i.e., 38 to 3F).

Typical sequence would be:

a. load hex buffer
b. CALL DGSTG
c. CALL ENREF (only done once)
d. EN I (only done once)

The next two routines are used as a part of the DGSTG routine but may be called separately.

DGOUT (Address 7ED)

Function: Moves character (in LED bit pattern form) into display buffer.

Reg. used: A, R0, R2

Reg. modified: A

Parameters expected:

A. R0: Location in display buffer (38-3F) in which character is to be inserted
B. R2: Character to be displayed.

HXOUT (Address 7F0)

Function: Decodes hex digit into LED bit pattern then performs DGOUT routine.

Reg. used: A, R0, R2

Reg. modified: A, R2

Parameters expected:

A. R2 LSN: Hex digit to be displayed
   MSN: Don't care
B. R0: Same as DGOUT.

```
ASM48 :F1:SYSCAL PAGELENGTH (52)

ISIS-II 8048 ASSEMBLER, V1.2

 LOC  OBJ        SEQ        SOURCE STATEMENT

                   0
                   1
                   2
                   3 ;PROGRAM NAME: 'SYSTEM CALLS' EXAMPLE FOR PROMPT 48
                   4
                   5 ;FUNCTION: PROGRAM READS THE KEYBOARD ON THE PROMPT 48
                   6 ;AND SHIFTS THE CHARACTER INTO A HEX BUFFER.  IT THEN
                   7 ;DISPLAYS THE HEX BUFFER ON THE 8 DIGIT LED DISPLAY.
                   8 ;ANY OF THE CONTROL KEYS WILL ACT AS THE DELIMITER
                   9 ;CAUSING THE DISPLAY TO BE BLANKED AND RESTARTING
                  10 ;THE ENTRY PROCESS.  IF MORE THAN 8 CHARACTERS ARE
                  11 ;ENTERED BEFORE THE DELIMITER THEY WILL BE SHIFTED
                  12 ;INTO THE LSD AND THE MSD WILL BE LOST.
                  13
                  14 ;ENTRY POINTS FOR MONITOR CALLS.
                  15
 07DC             16 BLK     EQU     7DCH            ;BLANK DISPLAY
 07F3             17 DGSTG   EQU     7F3H            ;DISPLAY ROUTINE
 07DF             18 ENREF   EQU     7DFH            ;ENABLE REFRESH
 07EA             19 KDBIN   EQU     7EAH            ;KEYBOARD ROUTINE
 07E2             20 REFS    EQU     7E2H            ;REFRESH ROUTINE
                  21
                  22 ;BUFFER AND MASK POINTERS
                  23
 0020             24 HBFPTR  EQU     20H             ;LSB OF HEX BUFFER
 003F             25 DBFPTR  EQU     3FH             ;MSB OF DISP BUFFER
 0036             26 DPMPTR  EQU     36H             ;DECIMAL POINT MASK
                  27
                  28 ;PROGRAM CONSTANTS
                  29
 0000             30 DPMASK  EQU     0               ;DECIMAL POINT MASK
 0004             31 HBUFL   EQU     4H              ;HEX BUFFER LENGTH
                  32
                  33 ;INTERRUPT VECTOR INITIALIZATION
                  34
 0000             35         ORG     0
 0000 E5          36         SEL     MB0
 0001 0409        37         JMP     MAIN
                  38
                  39 ;EXTERNAL INTERRUPT VECTOR
                  40
 0003 E4E2        41         JMP     REFS            ;REFR DONE ON INTERRUPT
                  42
                  43 ;TIMER INTERRUPT
                  44
```

```
ISIS-II 8048 ASSEMBLER, V1.2

 LOC  OBJ        SEQ        SOURCE STATEMENT

 0007             45         ORG    7
 0007 E4E2        46         JMP    REFS
                  47
                  48 ;INITIALIZATION OF DISPLAY, HEX BUF AND REFR MECH.
                  49
 0009 1435        50 MAIN:   CALL   CLRHX          ;HEX BUFR CLRD
 000B F4DC        51         CALL   BLK            ;BLANK DISP
 000D E5          52         SEL    MB0
 000E B836        53         MOV    R0,#DPMPTR     ;INIT DEC PT MASK
 0010 2300        54         MOV    A,#DPMASK
 0012 A0          55         MOV    @R0,A
 0013 18          56         INC    R0             ;POINT TO REFR CTR
 0014 2308        57         MOV    A,#8           ;INIT VAL OF REFR CTR
 0016 A0          58         MOV    @R0,A
 0017 F4DF        59         CALL   ENREF          ;HRDWR ENABLE OF REFR
 0019 E5          60         SEL    MB0
 001A 05          61         EN     I              ;REFRESH STARTS
                  62
                  63 ;GET KYBD CHAR AND CHECK IF DELIMITER
                  64
 001B F4EA        65 MAIN1:  CALL   KDBIN          ;GET CHAR
 001D E5          66         SEL    MB0
 001E 95          67         CPL    F0
 001F B628        68         JF0    HEXDG          ;YES-HEX DIGIT
                  69
                  70 ;CLEAR AND START OVER IF CONTROL CHAR
                  71
 0021 1435        72         CALL   CLRHX
 0023 F4DC        73         CALL   BLK            ;BLANK DISPLAY
 0025 E5          74         SEL    MB0
 0026 041B        75         JMP    MAIN1
                  76
                  77 ;HEX DIGIT - SHIFT INTO HEX BUFFER THEN DISPLAY BUFFER
                  78
 0028 143F        79 HEXDG:  CALL   HEXFL          ;SHIFT INTO HEX BUFFER
 002A BF08        80         MOV    R7,#HBUFL*2    ;NO OF DIG TO CONVERT
 002C B923        81         MOV    R1,#HBFPTR+3   ;MSB OF BUFFER
 002E B83F        82         MOV    R0,#DBFPTR
 0030 F4F3        83         CALL   DGSTG          ;CONVERT AND DISP BUFFER
 0032 E5          84         SEL    MB0
 0033 041B        85         JMP    MAIN1
                  86
                  87
                  88
                  89 ;SUBROUTINES
                  90
                  91 ;SUBROUTINE FUNCTION: CLEAR HEX BUFFER
```

```
ISIS-II 8048 ASSEMBLER, V1.2

 LOC  OBJ        SEQ        SOURCE STATEMENT

                   92 ;REG USED: A,R0,R2
                   93 ;REG MOD:  A,R0,R2
                   94
 0035 B820         95 CLRHX:  MOV     R0,#HBFPTR
 0037 BA04         96         MOV     R2,#HBUFL       ;LOOP COUNT
 0039 27           97         CLR     A
 003A A0           98 CLRHX1: MOV     @R0,A           ;CLEAR MEM LOC
 003B 18           99         INC     R0
 003C EA3A        100         DJNZ    R2,CLRHX1
 003E 83          101         RET
                  102
                  103
                  104 ;SUBROUTINE FUNCTION: SHIFT ACCUM LSN INTO HEX BUFFER
                  105 ;REG USED: A,R1,R2
                  106 ;REG MOD: A,R1,R2
                  107
 003F B920        108 HEXFL:  MOV     R1,#HBFPTR
 0041 BA04        109         MOV     R2,#HBUFL       ;LOOP COUNT
 0043 21          110 HEXFL1: XCH     A,@R1           ;GET LOW DIGIT PAIR
 0044 47          111         SWAP    A               ;SWAP NIBBLES
 0045 31          112         XCHD    A,@R1           ;INSERT A LSN
 0046 21          113         XCH     A,@R1           ;RESTORE BYTE
 0047 19          114         INC     R1
 0048 EA43        115         DJNZ    R2,HEXFL1
 004A 83          116         RET
                  117
                  118         END

USER SYMBOLS

BLK    07DC    CLRHX  0035    CLRHX1 003A    DBFPTR 003F    DGSTG  07F3    DPMASK 0000    DPMPTR 0036
ENREF  07DF    HBFPTR 0020    HBUFL  0004    HEXDG  0028    HEXFL  003F    HEXFL1 0043    KDBIN  07EA
MAIN   0009    MAIN1  001B    REFS   07E2

ASSEMBLY COMPLETE, NO ERROR(S)
```

APPENDIX C
PROGRAMMING EXAMPLE: STOPWATCH

Problem Definition

Assume that you wish to write a program that will allow Prompt-48 to function as a stopwatch. As always, the first step in accomplishing this task is to define exactly what you want the program to do. At a minimum, a stopwatch must be able to stop, clear, start, and display the contents of a timer. The timer must have a resolution suited to the intended use of the stopwatch. Since the purpose of this program is to illustrate programming techniques, you can be arbitrary and give the timer a resolution of 1/100 second. Let's go further and add two more things for the stopwatch to do: freeze the display at the current value of the timer while allowing the timer to continue running; release the display to show the contents of the timer. This will allow the stopwatch to function as a "lap counter."

Now that you have a slightly better idea of what the program is to do, you can take a stab at dividing it into submodules. The first and most important submodule is the module which decides what is to be done on the basis of keys pressed by the user. This module can be thought of as the executive, as it is given control of the program at the start, and control invariably returns to it after each command is executed. We will call this module the User Control Functions (Commands) module.

The User Control Functions (Commands) module must have at least indirect access to the LED display on the Prompt-48 panel. The module which contains the routines to take care of the display will be called the Display Functions module.

The Data Functions module will give the User Control Functions module the means to read the panel keyboard, clear the variable TIME, and correctly add 1 to the minutes, seconds, and hundredths of seconds of TIME.

The last major submodule of the stopwatch program is the Timer Control Functions module. This module contains the routines which start, stop, and reset for 1/100 second to overflow that MCS-48 Timer register.

The major submodules and their breakdowns into these more basic tasks are shown in Figure C-1.

Modular Interfaces

Now that we have a general structure for the stopwatch program, we can design the modular interfaces, or the ways in which the program modules communicate with one another.

The User Control Functions module must request data and tasks to be performed of the other three major submodules. The easiest way, to establish the simple communications necessary, is to assign one or more registers to hold instructions or data passing from one module to another. For example, the User Control Functions module, when requesting the Data Functions module to add 1 to TIME, might place a hexadecimal 02 in the Accumulator before passing control to the Data Functions module. The Data Functions module would then examine the Accumulator to see what was being requested of it, having been written with the knowledge that hex 02 means "add 1 to TIME." Any combination of Registers, user Flags, or Data Memory locations can be used in this way to accomplish a given programming task.

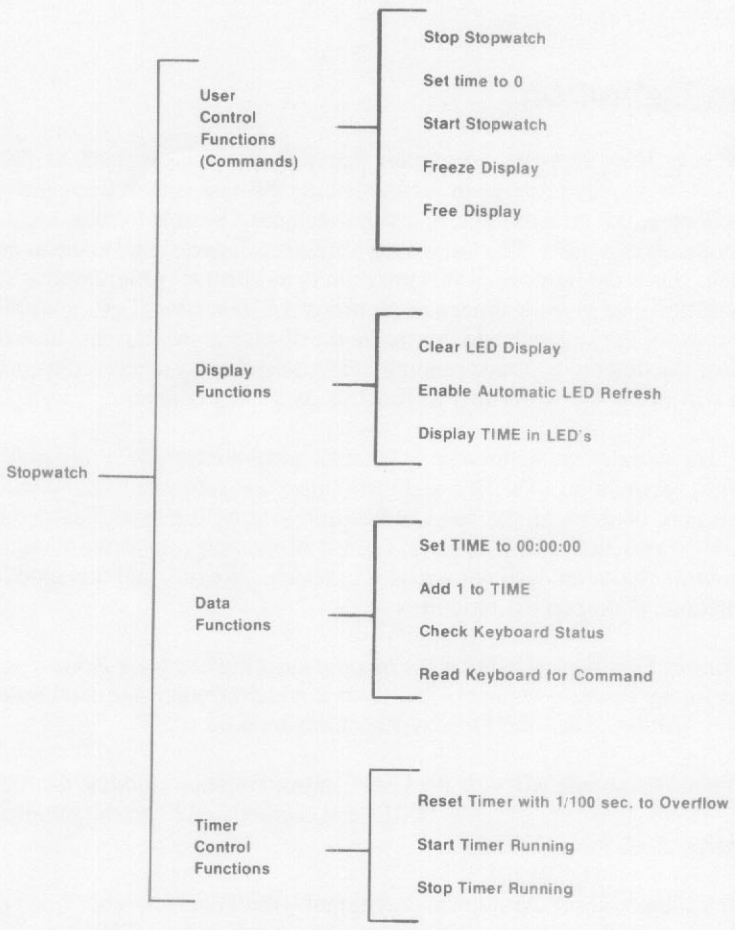


Figure C-1: Stopwatch Program Structure

This concept can also be used in another way, called "switching." A switch is appropriate in the stopwatch program for the purpose of freezing the LED display at the value of TIME when the freeze command is received from the keyboard. The Display TIME in LED's submodule of Display Functions module needs to know whether or not the display is "frozen." This information can be stored in a bit of a Register or Data Memory location, or in a user Flag, by the module deciding to freeze or unfreeze the LED's, and later examined by the Display TIME in LED's module to see whether a new value should be displayed. This switch amounts to the ability to turn on or turn off the automatic update of TIME in the LED's.

These concepts are used in the actual stopwatch program, and are commented upon in the program listing.

Display Functions

Control of the Prompt-48 panel LED's is handled through the use of various System Calls (see Appendix B). These System Calls make it extremely easy to display whatever numerical data we wish in the LED's.

The System Calls used for display are BLK, ENREF, and HXOUT, each of whose use is described in Appendix B. It should be noted that the actual LED refresh is handled on interrupt from Program Memory location 03. This interrupt is automatically generated by the Prompt-48 hardware after the ENREF System Call is used.

Data Functions

TIME is held in three consecutive bytes of Register memory, representing minutes, seconds, and hundredths of seconds. These values are in decimal, requiring the capability to add in decimal arithmetic instead of hexadecimal. This is easily accomplished with the DA A (Decimal Adjust Accumulator) instruction, as described in the MCS-48 Assembly Language Manual. A problem still remains in that seconds can never exceed 59_{10}. The Add 1 to TIME module must examine the result of adding 1 to seconds in the event of a carry from hundredths. If the seconds portion of TIME is equal to 60, it is set to 00 and 1 is added to minutes. This is called modulo arithmetic, with seconds being maintained mod(ulo) 60. Decimal notation is mod 10, as no single digit is ever allowed to exceed 9.

The keyboard status is checked, and the keyboard read, through the use of two System Calls, KBST and KBIN.

Timer Control

The value loaded into the Timer should result in a 1/100 second delay to when the Timer overflows and sets TF (Timer Flag) = 1. To determine what the proper value is, we note that the Timer gets incremented every 32 instruction cycles, or 1/480 the clock crystal frequency. The standard MCS-48 Chip-Computer runs at a maximum frequency of 3 MHz, so the Timer will increment 1/480 × 3 MHz = 6250 times a second, or 62.5 times every 1/100 second. The value to be loaded into the Timer should therefore allow the Timer to increment 62 times before overflow occurs. This value is computed by taking the hexadecimal equivalent of -62_{10}, which is $C2_{16}$. This will equal 00_{16} plus a carry (overflow, or TF = 1) when it has been incremented 62_{10} times. For maximum accuracy, the .5 in 62.5 must also be taken into account. This is accomplished by delaying ½ of a Timer increment (16 instruction cycles) between overflow of the Timer and the next load of −62.

```
asm48 :f1:stpwch

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    1

  LOC  OBJ          SEQ          SOURCE STATEMENT

                      1 $print(:TO:)
                      2 $pagewidth(80) pagelength(66) noobject
                      3 ;*************************************************************
                      4 ;
                      5 ; Prompt-48 Programming Example:  Stopwatch
                      6 ;
                      7 ; This program is intended as an illustration of medium
                      8 ; complexity programming techniques for the MCS-48
                      9 ; family of Chip-Computers, to be run in Prompt-48.
                     10 ;
                     11 ; The complete and verified program will allow the user,
                     12 ; if desired, to observe the Prompt-48 computer in
                     13 ; action almost immediately upon delivery, and will
                     14 ; give him or her a practice example for EPROM burning
                     15 ; as well.  Without needing to know anything of the
                     16 ; MCS-48 assembly language in which Stopwatch is
                     17 ; written, the user can simply enter into Program
                     18 ; Memory the sequence of (hexadecimal) object code
                     19 ; which appears in the second (OBJ) column of this
                     20 ; listing.  (Refer to Appendix A, "A Familiarization
                     21 ; Exercise", for key sequences used for entering and
                     22 ; examination of Program Memory contents.)  Note that
                     23 ; there are two jumps in the normal sequence of
                     24 ; addresses (found in the LOC column) in the program:
                     25 ; 4H to 100H, and 1D5H to 200H.  These jumps are
                     26 ; made as a programming convenience arising from the
                     27 ; MCS-48 Program Memory paging feature.  Once the
                     28 ; whole object file is correctly entered, and the
                     29 ; correct Access Mode is specified, the Stopwatch
                     30 ; will be operational.  Refer to the listing for
                     31 ; program use and command key definitions.
                     32 ;
                     33 ; If RAM Program Memory is used, use Access Mode 2.  If
                     34 ; the program is burned into an 8748, use Access Mode 5.
                     35 ;
                     36 ;*************************************************************
                     37 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    2

  LOC  OBJ          SEQ          SOURCE STATEMENT

                      36 ;
                      37 ; Program Structure in Hierarchical Form:
                      40 ;
                      41 ; Stopwatch
                      42 ;
                      43 ; 1. User Control Functions (Executive Section)
                      44 ;    1a. Start Stopwatch ([GO] key)
                      45 ;    1b. Stop Stopwatch ([BREAK] key)
                      46 ;    1c. Freeze Display ([EXAM] key)
                      47 ;    1d. Free Display ([NEXT] key)
                      48 ;    1e. Stop Stopwatch and Clear TIME ([END] key)
                      49 ;
                      50 ; 2. Display Functions (disitn)
                      51 ;    2a. Clear LED display
                      52 ;    2b. Enable Automatic LED Refresh
                      53 ;    2c. Display TIME in LEDs
                      54 ;
                      55 ; 3. Data Functions (datafn)
                      56 ;    3a. Clear TIME to 0:00.00
                      57 ;    3b. Add 1 to TIME
                      58 ;    3c. Return Keyboard Status
                      59 ;    3d. Return Key Data
                      60 ;
                      61 ; 4. Timer Reset Routine
                      62 ;
                      63 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0          PAGE    3

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     64 ;
                     65 ; symbol declarations
                     66 ;
                     67
                     68 ; disftn symbols
                     69
  0001               70 clrdsp  equ     1        ;clear led display command
  0002               71 enrfsh  equ     2        ;enable led refresh command
  0003               72 distim  equ     3        ;display TIME in leds command
                     73
                     74 ; datain symbols
                     75
  0001               76 clrtim  equ     1        ;clear TIME command
  0002               77 inctim  equ     2        ;add 1 to TIME command
  0003               78 keyst   equ     3        ;keyboard status request
  0004               79 kydata  equ     4        ;keyboard data request
                     80 ;
                     81 ; system call addresses
                     82 ;
  07E4               83 kbst    equ     7e4h     ;get keyboard status
  07E7               84 kbin    equ     7e7h     ;get keyboard data
  07E2               85 refs    equ     7e2h     ;refresh led display
                     86 ;                         (on interrupt)
  07DC               87 blk     equ     7dch     ;blank led display
  07DF               88 enref   equ     7dfh     ;enable refresh interrupt
  07F3               89 dgstg   equ     7f3h     ;display multiple hex digits
  07F0               90 hxout   equ     7f0h     ;decode and display hex digit
                     91 ;
                     92 ; data register assignments
                     93 ;
  0003               94 freeze  equ     3        ;freeze switch in r3
  0020               95 time    equ     20h      ;TIME in r20-r22
                     96 ;
                     97 ; command key symbol assignments
                     98 ;
  0015               99 starts  equ     15h      ;[GO] = start command
  0012              100 stops   equ     12h      ;[BREAK] = stop command
  0014              101 freezs  equ     14h      ;[EXAM] = freeze command
  0016              102 frees   equ     16h      ;[NEXT] = free command
  0017              103 stpclr  equ     17h      ;[END] = stop and clear
                    104 ;
                    105 ; miscellaneous constants and addresses
                    106 ;
  0036              107 dpmsk   equ     36h      ;led dec. point mask address
  0000              108 dpoff   equ     0        ;decimal points off pattern
  0064              109 dbnce   equ     100      ;debounce loop length
                    110
                    111 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    4

  LOC  OBJ          SEQ          SOURCE STATEMENT

  0000                112          org     0          ;starting address of program
                      113 ;
                      114 ; start of program
                      115 ;
  0000 B5             116 start:   sel     mb0        ;select program mem bank 0
  0001 2400           117          jmp     exec       ;jump to executive section
                      118 ;
                      119 ; interrupt vectors
                      120 ;
  0003                121          org     3          ;led refresh vector address
  0003 B4E2           122          jmp     refs       ;refresh leds
                      123 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0          PAGE    5

 LOC  OBJ          SEQ          SOURCE STATEMENT

                    124 ;*******************************************************
                    125 ;
                    126 ; executive section:
                    127 ;
                    128 ;       This section controls the overall program
                    129 ;       execution.  It communicates with the following
                    130 ;       modules:
                    131 ;
                    132 ;       1) disftn - display functions.  Clears,
                    133 ;          enables for refresh, or displays TIME in
                    134 ;          leds.
                    135 ;
                    136 ;       2) datafn - data functions.  Clears TIME, adds
                    137 ;          1 to TIME, checks keyboard status, or reads
                    138 ;          keyboard for command input.
                    139 ;
                    140 ;       3) tmrrst - timer reset.  resets MCS-48
                    141 ;          timer for 1/100 sec. to overflow.
                    142 ;
                    143 ;*******************************************************
                    144 ;
                    145 ; start of executive: first, initialize timer, TIME,
                    146 ;       and display.
                    147 ;
 0100               148          org     100h    ;start at page 1
 0100 65            149 exec:    stop    tcnt    ;stop timer
 0101 2300          150          mov     a,#0    ;clear a
 0103 62            151          mov     t,a     ;clear timer
 0104 545D          152          call    tmrrst  ;1/100 sec delay in timer
 0106 B803          153          mov     r0,#freeze ;point to freeze switch
 0108 B000          154          mov     @r0,#0  ;unfreeze display
 010A BF01          155          mov     r7,#clrtim ;clear TIME command
 010C 5400          156          call    datafn  ;data functions module
 010E BF03          157          mov     r7,#distim ;display TIME command
 0110 347E          158          call    disftn  ;display funcions module
 0112 BF02          159          mov     r7,#enrfsh ;enable led refresh command
 0114 347E          160          call    disftn  ;display functions module
                    161 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0          PAGE    6

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     162 ;
                     163 ; now wait for input commands.
                     164 ;
                     165 ; the commands are:
                     166 ;       [GO] - start stopwatch
                     167 ;       [BREAK] - stop stopwatch
                     168 ;       [EXAM] - freeze display at present TIME
                     169 ;       [NEXT] - free display to follow TIME
                     170 ;       [END] - stop stopwatch and clear TIME
                     171 ;
                     172 ;*****************************************************
                     173 ;
                     174 ; monitor loop.  This part of the executive waits
                     175 ;       until datain indicates a key is being
                     176 ;       pressed, or the timer overflows.
                     177 ;
                     178 ;       If a key is pressed, the command (if defined)
                     179 ;       is processed and the executive returns
                     180 ;       to the monitor loop.
                     181 ;
                     182 ;       If the timer overflows, one is added
                     183 ;       to TIME.  The display is then updated if
                     184 ;       FREEZE = 0, and the executive returns to
                     185 ;       the monitor loop.
                     186 ;
0116 3468            187 monitr: call    uptim   ;update TIME if necessary
0118 BF03            188         mov     r7,#keyst ;keyboard status request
011A 5400            189         call    datain  ;data functions module
011C E616            190         jnc     monitr  ;loop if no key pressed
                     191 ;
                     192 ; key being pressed: input command for processing.
                     193 ;
011E BF04            194         mov     r7,#kydata ;keyboard data request
0120 5400            195         call    datain  ;data functions module
                     196 ;
                     197 ; start stopwatch command?
                     198 ;
0122 FF              199         mov     a,r7    ;key data in a
0123 03EE            200         add     a,#-starts ;start command?
0125 962A            201         jnz     next1   ;jump if not
                     202
0127 55              203         strt    t       ;start timer
0128 C45E            204         jmp     endcom  ;end of command processing
                     205 ;
                     206 ; not start:  stop stopwatch command?
                     207 ;
012A FF              208 next1:  mov     a,r7    ;key data in a
012B 03EB            209         add     a,#-stops ;stop command?
012D 9632            210         jnz     next2   ;jump if not
                     211
012F 65              212         stop    tcnt    ;stop timer
0130 C45E            213         jmp     endcom  ;end of command processing
                     214
                     215 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0          PAGE    7

  LOC  OBJ          SEQ          SOURCE STATEMENT

                    216 ;
                    217 ; not stop:  freeze command?
                    218 ;
  0132 FF           219 next2:  mov     a,r7     ;key data in a
  0133 03EC         220         add     a,#-freezs ;freeze command?
  0135 9642         221         jnz     next3    ;jump if not
                    222
  0137 B803         223         mov     r0,#freeze ;point to freeze switch
  0139 2301         224         mov     a,#1     ;"freeze display"
  013B A0           225         mov     @r0,a    ;set freeze switch
  013C BF03         226         mov     r7,#distim ;display TIME command
  013E 547B         227         call    disitn   ;display functions module
  0140 245B         228         jmp     endcom   ;end of command processing
                    229 ;
                    230 ; not freeze:  free command?
                    231 ;
  0142 FF           232 next3:  mov     a,r7     ;key data in a
  0143 03EA         233         add     a,#-frees ;free command?
  0145 9648         234         jnz     next4    ;jump if not
                    235
  0147 B803         236         mov     r0,#freeze ;point to freeze switch
  0149 2300         237         mov     a,#0     ;"free display"
  014B A0           238         mov     @r0,a    ;clear freeze switch
  014C 245B         239         jmp     endcom   ;end of command processing
                    240 ;
                    241 ; not free:  stop and clear command, or undefined key.
                    242 ;
  014E FF           243 next4:  mov     a,r7     ;key data in a
  014F 03E9         244         add     a,#-stpclr ;stop and clear command?
  0151 965B         245         jnz     endcom   ;jump, undefined if not
                    246
  0153 65           247         stop    tcnt     ;stop timer
  0154 5456         248         call    tmrrst   ;stop and reset timer
  0156 BF01         249         mov     r7,#clrtim ;clear time command
  0158 5400         250         call    datain   ;data functions module
  015A BF03         251         mov     r7,#distim ;display TIME command
  015C 547B         252         call    disitn   ;display functions module
                    253
                    254 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    8

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     255 ;
                     256 ; end of command processing:  wait for the key
                     257 ;       to be released, then return to the monitor
                     258 ;       loop.
                     259 ;
  015E 3466          260 endcom: call    uptim   ;update TIME if necessary
  0160 BF03          261         mov     r7,#keyst ;keyboard status request
  0162 5400          262         call    datain  ;data functions module
  0164 F65E          263         jc      endcom  ;loop until key released
                     264
  0166 2416          265         jmp     monitr  ;return to monitor loop
                     266
                     267 ;
                     268 ; uptim - update TIME if necessary.  Subroutine
                     269 ;       to check the status of TF (Timer Flag), and
                     270 ;       add 1 to TIME if TF = 1.  Datain (Data functions)
                     271 ;       module is used to add 1 to TIME.
                     272 ;
                     273 ; reg modified: a, r0, r2, r7
                     274 ;
  0168 1660          275 uptim:  jt1     next5   ;skip ahead if TF = 1
                     276
  016A 247D          277         jmp     uptxit  ;jump to exit if TF = 0
                     278
  016C 545D          279 next5:  call    tmrrst  ;1/100 sec delay for timer
  016E BF02          280         mov     r7,#inctim ;add 1 to TIME command
  0170 5400          281         call    datain  ;data functions module
  0172 B803          282         mov     r0,#freeze ;point to freeze switch
  0174 F0            283         mov     a,@r0   ;freeze switch in a
  0175 D300          284         xrl     a,#0    ;freeze switch = 0?
  0177 967D          285         jnz     uptxit  ;jump to exit if not
                     286
  0179 BF03          287         mov     r7,#distim ;display TIME command
  017B 347B          288         call    disftn  ;display functions module
                     289
  017D 83            290 uptxit: ret             ;exit uptim
                     291
                     292 ;
                     293 ; end of executive section
                     294 ;
                     295
                     296 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    9

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     297 ;*******************************************************
                     298 ;
                     299 ; disitn - display functions module.  Three commands
                     300 ;       are executed by this module:
                     301 ;
                     302 ;       1) clrdsp - clear led display
                     303 ;          reg modified: a, r0, r7
                     304 ;
                     305 ;       2) enrish - enable automatic led refresh
                     306 ;          reg modified: a, r0, r7, p2
                     307 ;
                     308 ;       3) distim - display time in leds
                     309 ;          reg modified: a, r0, r1, r2, r7
                     310 ;
                     311 ;
                     312 ; The command is received in r7.  The module is called
                     313 ;       as a subroutine.
                     314 ;
                     315 ;*******************************************************
                     316
017E FF              317 disitn: mov     a,r7    ;command in a
                     318
                     319 ;
                     320 ; clrdsp command?
                     321 ;
017F 03FF            322         add     a,#-clrdsp ;clrdsp command?
0181 9686            323         jnz     nextb   ;jump if not
                     324
0183 F4DC            325         call    blk     ;blank leds sys call
0185 E5              326         sel     mb0     ;fix program mem bank
0186 24B5            327         jmp     dspxit  ;jump to exit
                     328 ;
                     329 ; not clrdsp:  enrish command?
                     330 ;
0188 FF              331 nextb:  mov     a,r7    ;command in a
0189 03FE            332         add     a,#-enrish ;enrish command?
018B 969A            333         jnz     next7   ;jump if not
                     334
018D B830            335         mov     r0,#dpmsk ;point to decimal point mask
018F B000            336         mov     @r0,#dpoff ;decimal points off
0191 18              337         inc     r0      ;point to refresh char pointer
0192 B008            338         mov     @r0,#8  ;led 8 first
0194 F4DF            339         call    enref   ;enable led refresh sys call
0196 E5              340         sel     mb0     ;fix program mem bank
0197 05              341         en      i       ;enable refresh interrupts
0198 24B5            342         jmp     dspxit  ;jump to exit
                     343 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE   10

  LOC  OBJ          SEQ          SOURCE STATEMENT

                    344 ;
                    345 ; not enrfsh:  either distim or undefined command
                    346 ;
  019A FF           347 next7:  mov     a,r7     ;command in a
  019B 03FD         348         add     a,#-distim ;undefined command?
  019D 96D5         349         jnz     dspxit   ;jump to exit if so
                    350
                    351 ; display minutes
                    352
  019F B83E         353         mov     r0,#3eh ;led address of minutes
  01A1 B920         354         mov     r1,#time ;minutes portion of TIME
  01A3 BF02         355         mov     r7,#2    ;2 digits to be displayed
  01A5 F4F3         356         call    dgstg    ;convert and display
  01A7 E5           357         sel     mb0      ;fix program mem bank
                    358
                    359 ; display seconds
                    360
  01A8 B83B         361         mov     r0,#3bh ;msd led destination
  01AA B921         362         mov     r1,#time+1 ;seconds portion of TIME
  01AC F1           363         mov     a,@r1    ;move seconds to a
  01AD 47           364         swap    a        ;msd in ls nibble
  01AE AA           365         mov     r2,a     ;hex display data - msd
  01AF F4F0         366         call    hxout    ;display seconds msd
  01B1 E5           367         sel     mb0      ;fix program mem bank
  01B2 F1           368         mov     a,@r1    ;move seconds to a
  01B3 AA           369         mov     r2,a     ;hex display data - lsd
  01B4 C8           370         dec     r0       ;lsd led destination
  01B5 F4F0         371         call    hxout    ;display seconds lsd
  01B7 E5           372         sel     mb0      ;fix program memory bank
                    373
                    374 ; display hundredths of seconds
                    375
  01B8 B83A         376         mov     r0,#3ah ;seconds lsd led address
  01BA F0           377         mov     a,@r0    ;led code in a
  01BB 537F         378         anl     a,#7fh   ;decimal point on
  01BD A0           379         mov     @r0,a    ;replace in led buffer
  01BE C8           380         dec     r0       ;msd led destination
  01BF B922         381         mov     r1,#time+2 ;hundredths portion of TIME
  01C1 F1           382         mov     a,@r1    ;hundredths in a
  01C2 47           383         swap    a        ;msd in ls nibble
  01C3 AA           384         mov     r2,a     ;hex display data - msd
  01C4 F4F0         385         call    hxout    ;display hundredths msd
  01C6 E5           386         sel     mb0      ;fix program mem bank
  01C7 C8           387         dec     r0       ;lsd led destination
  01C8 F1           388         mov     a,@r1    ;lsd hex data
  01C9 AA           389         mov     r2,a     ;hex display data - lsd
  01CA F4F0         390         call    hxout    ;display hundredths lsd
  01CC E5           391         sel     mb0      ;fix program mem bank
                    392
                    393 ; clear unused leds
                    394
  01CD B83F         395         mov     r0,#3fh ;leftmost led
  01CF B0FF         396         mov     @r0,#0ffh ;clear leftmost led
  01D1 B83C         397         mov     r0,#3ch ;led between minutes, seconds
  01D3 B0FF         398         mov     @r0,#0ffh ;clear it
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                    PAGE    11

 LOC  OBJ          SEQ          SOURCE STATEMENT

                    399
                    400 ;
                    401 ; aisitn exit point
                    402 ;
                    403
 01B5 83            404 aspxit: ret                ;exit aisitn
                    405
                    406 ;
                    407 ; end of aisitn module
                    408 ;
                    409
                    410
                    411 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE   12

  LOC  OBJ          SEQ          SOURCE STATEMENT

  0200               412          org     200h    ;new program mem page
                     413 ;*****************************************************
                     414 ;
                     415 ; datafn - data funcions module.  Four commands
                     416 ;       are executed by this module:
                     417 ;
                     418 ;       1) clrtim - clear TIME.  Sets TIME to 00:00:00.
                     419 ;          reg modified:  a, r0, r7
                     420 ;
                     421 ;       2) inctim - increment TIME.  Adds 1 to TIME.
                     422 ;          reg modified:  a, r0, r7
                     423 ;
                     424 ;       3) keyst - key status.  Determines whether
                     425 ;          a key is being pressed.
                     426 ;          output: c = 1 if a key is pressed
                     427 ;                  c = 0 if no key pressed
                     428 ;          reg modified:  a, r0, r7, p2, c
                     429 ;
                     430 ;       4) kydata - key data.  Determines which
                     431 ;          key is being pressed.
                     432 ;          output: r7 = key value
                     433 ;          reg modified: a, r0, r6, r7, p2, 10, c
                     434 ;
                     435 ;
                     436 ; The module is called as a subroutine with the
                     437 ;       command in r7.
                     438 ;
                     439 ;*****************************************************
                     440
  0200 FF            441 datain: mov     a,r7    ;command in a
                     442
                     443 ;
                     444 ; clrtim command?
                     445 ;
  0201 03FF          446         add     a,#-clrtim ;clrtim command?
  0203 9613          447         jnz     next9   ;jump if not
                     448
  0205 2300          449         mov     a,#0    ;put zero in a
  0207 B820          450         mov     r0,#time ;point to TIME
  0209 BF03          451         mov     r7,#3   ;loop counter
                     452
  020B A0            453 cltlp:  mov     @r0,a   ;clear one byte of TIME
  020C 18            454         inc     r0      ;point to next byte
  020D EF0B          455         djnz    r7,cltlp ;loop till TIME = 0
                     456
  020F 1611          457         jti     next12  ;clear timer flag, to prevent TI
                         ME
                     458 ;                       ;...from incrementing after clea
                         r
                     459
  0211 4450          460 next12: jmp     dtaxit  ;jump to exit
                     461
                     462 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE   13

  LOC  OBJ          SEQ          SOURCE STATEMENT

                    463 ;
                    464 ; not clrtim:  inctim command?
                    465 ;
  0213 FF           466 nexty:  mov     a,r7     ;command in a
  0214 03FB         467         add     a,#-inctim ;inctim command?
  0216 9635         468         jnz     next10   ;jump if not
                    469
                    470 ; add 1 to hundredths
                    471
  0218 B022         472         mov     r0,#time+2 ;point to hundredths
  021A F0           473         mov     a,@r0    ;move data to a
  021B 0301         474         add     a,#1     ;add 1 to hundredths data
  021D 57           475         da      a        ;decimal adjust
  021E A0           476         mov     @r0,a    ;update hundredths
  021F E65C         477         jnc     dtaxit   ;exit if no carry
                    478
                    479 ; carry into seconds
                    480
  0221 C8           481         dec     r0       ;point to seconds
  0222 F0           482         mov     a,@r0    ;move data to a
  0223 0301         483         add     a,#1     ;increment seconds
  0225 57           484         da      a        ;decimal adjust
  0226 A0           485         mov     @r0,a    ;update seconds
  0227 03A0         486         add     a,#-60h  ;mod 60 overflow test
  0229 E65C         487         jnc     dtaxit   ;exit if no overflow
                    488
  022B B000         489         mov     @r0,#0   ;60 becomes 0
                    490
                    491 ; carry into minutes
                    492
  022D C8           493         dec     r0       ;point to minutes
  022E F0           494         mov     a,@r0    ;move data to a
  022F 0301         495         add     a,#1     ;increment minutes
  0231 57           496         da      a        ;decimal adjust
  0232 A0           497         mov     @r0,a    ;update minutes
  0233 445C         498         jmp     dtaxit   ;jump to exit
                    499
                    500 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE   14

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     501 ;
                     502 ; not inctim:  keyst command?
                     503 ;
  0235 FF            504 next10: mov     a,r7     ;command in a
  0236 03F6          505         add     a,#-keyst ;keyst command?
  0238 9649          506         jnz     next11   ;jump if not
                     507
  023A EB04          508         mov     r6,#dbnce ;debounce loop counter
                     509
  023C F44B          510 dbnclp: call    kbst     ;get key status
  023E B5            511         sel     mb0      ;fix program mem bank
  023F E65C          512         jnc     dtaxit   ;exit if no key pressed
  0241 3468          513         call    uptim    ;update time if necessary
  0243 EB3C          514         djnz    r6,dbnclp ;loop till debounce done
                     515
  0245 BF02          516         mov     r7,#enrish ;reenable refresh of leds
  0247 347E          517         call    disitn   ;...via disitn module
  0249 97            518         clr     c        ;ensure carry is
  024A A7            519         cpl     c        ;...still set
  024B 4456          520         jmp     dtaxit   ;jump to exit
                     521 ;
                     522 ; not keyst:  kydata command?
                     523 ;
  024D FF            524 next11: mov     a,r7     ;command in a
  024E 03F6          525         add     a,#-kydata ;kydata command?
  0250 9656          526         jnz     dtaxit   ;(undefined) exit if not
                     527
  0252 F4E7          528         call    kbin     ;get key data
  0254 B5            529         sel     mb0      ;fix program mem bank
  0255 AE            530         mov     r6,a     ;save key data in r6
  0256 BF02          531         mov     r7,#enrish ;reenable led refresh
  0258 347E          532         call    disitn   ;...via disitn module
  025A FE            533         mov     a,r6     ;get key data
  025B AF            534         mov     r7,a     ;data in output register
                     535
                     536 ;
                     537 ; datafn exit point
                     538 ;
                     539
  025C 83            540 dtaxit: ret              ;exit datafn module
                     541
                     542 ;
                     543 ; end datafn module
                     544 ;
                     545
                     546
                     547 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    15

  LOC  OBJ          SEQ          SOURCE STATEMENT

                     548 ;*************************************************
                     549 ;
                     550 ; tmrrst - timer reset module.  This routine adds to
                     551 ;        the contents of the timer -62 (decimal).
                     552 ;        It is done this way because the timer may
                     553 ;        have been incrementing during an interrupt
                     554 ;        routine between the detection of Tr = 1
                     555 ;        and the actual reset.
                     556 ;
                     557 ;        -62 decimal is the value closest to 1/100
                     558 ;        second delay for a 3 Mhz clock frequency.
                     559 ;
                     560 ;        A 16 instruction cycle (1/2 timer increment)
                     561 ;        delay should be added at the beginning of the
                     562 ;        routine, with the timer stopped, for maximum
                     563 ;        accuracy.
                     564 ;
                     565 ;*************************************************
                     566
0259 42              567 tmrrst: mov     a,t         ;timer data to a
025A 0362            568         add     a,#-62      ;counter for 1/100
                     569 ;                            second delay
025C 62              570         mov     t,a         ;reset timer
025D 63              571         ret                 ;exit tmrrst
                     572
                     573 ;
                     574 ; end of tmrrst module
                     575 ;
                     576
                     577
                     578 $eject
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    16

  LOC  OBJ          SEQ          SOURCE STATEMENT

  0000               579          end     0       ;end of stopwatch program

USER SYMBOLS
BLK     07DC    CLRDSP 0001    CLRTIM 0001    CLTLP  020B    DATAPR 0200
DBNCE   0064    DBNCLP 023C    DGSTG  07F3    DISPTM 017E    DISTIM 0003
DPMSK   0036    DPOFF  0000    DSPAIT 01D5    DTAXIT 025C    ENDCOM 015E
ERRER   07DF    ERRFSH 0002    EXEC   0100    FREES  0016    FREEZE 0003
FREEZS  0014    INCOUT 07FC    INCTIM 0002    KEIN   07E7    KEST   07E4
KEYST   0003    KYDATA 0004    MONITR 0110    NEXT1  012A    NEXT10 0235
NEXT11  024D    NEXT12 0211    NEXT2  0132    NEXT3  0142    NEXT4  014E
NEXT5   016C    NEXT6  0188    NEXT7  019A    NEXT9  0213    RERS   07E2
START   0000    STARTS 0015    STOPS  0012    STPCLR 0017    TIME   0020
TMRRST  025D    UPTIM  0188    UPTXIT 017D

ASSEMBLY COMPLETE,  NO ERRORS
```

APPENDIX D
HEXADECIMAL OBJECT FILE FORMAT

Hexadecimal object code format is an ASCII representation of program memory, expressed as a series of hexadecimal digits. These are blocked into records, each of which contains the record length, type, memory load address, and checksum, in addition to data.

Frame 0. Record Mark. The ASCII representation of a colon ($3A_{16}$) is used to signal the start of a record.

Frames 1 and 2. Record length in hexadecimal. This is the count of the actual data bytes in the record. Frame 1 contains the high-order digit of the count, and frame 2 contains the low-order digit. A record length of zero indicates end of file.

Frames 3 to 6. Load address. The four-character starting address at which the following data will be loaded. The high-order digit of the load address is in frame 3, and the low-order digit is in frame 6. The first data byte is stored in the location indicated by the load address. Successive data bytes are stored in successive memory locations.

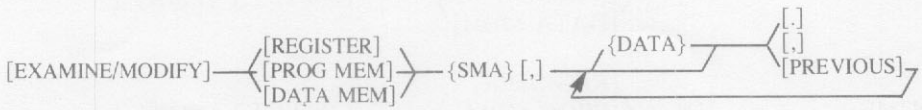
Frames 7 and 8. Record type. A two-digit code in this field specifies the type of this record. The high-order digit of this code is located in frame 7. Currently, all data records are type 0. End-of-file records may be type 0 or type 1. In either case they are distinguished by a zero record length field (see above).

Frames 9 to 9+2*(record length)−1. Data. Each 8-bit memory word is represented by two frames containing ASCII characters 0-9, A-F, which represent a hexadecimal value between 0 and FF (0 and 255_{10}). The high-order digit of each byte is located in the first frame of each pair.

Frames 9+2*(record length) to 9+2*(record length)+1. Checksum. The checksum is the negative of the sum of all 8-bit bytes in the record, evaluated modulo 256. The sum of all bytes in the record (including the checksum) should be zero.

APPENDIX E
COMMAND/FUNCTION SUMMARY

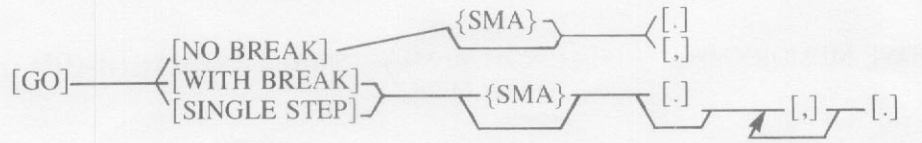
Commands



MCS 48 processors have 64 bytes of register memory numbered 0-$3F_{16}$. PROMPT 48 allows access to other "register" locations via [EXAMINE/MODIFY] [REGISTER].

Number	Location	Format
40	ACCUMULATOR	
41	TIMER	
42	PSW	C | AC | F0 | RB | F1 | S_2 | S_1 | S_0
43	PCL	
44	PCH	
45	PORT 0 (BUS)	READ-ONLY
46	PORT 1	READ-ONLY
47	PORT 2	
48	MISC	Counter Run | Timer Run | Timer Flag | Nested Fr Int | Will En Int | Mem Bank | T1 | T0

PROMPT 48 provides 256 bytes of data memory numbered 0-FF_{16}.



Ensure you have selected the correct access code, P2 MAP and LSN P2 contents before running programs.

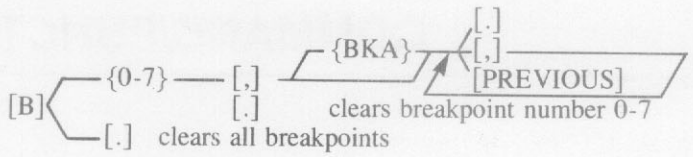
{SMA} Starting register/memory address
{DATA} Data

Functions

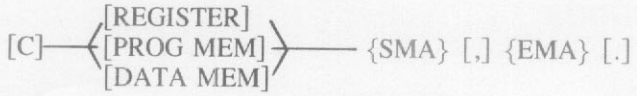
ACCESS [A] {0-5} [.]

Access Code	Program Memory	System I/O and System Calls	Expansion Memory and I/O	OUTL Port 0
0	WRITABLE (RAM)	no	no	yes
1	WRITABLE (RAM)	no	yes	no
2	WRITABLE (RAM)	yes	no	no
3	READ ONLY (ON CHIP)	no	no	yes
4	READ ONLY (ON CHIP)	no	yes	no
5	READ ONLY (ON CHIP)	yes	no	no

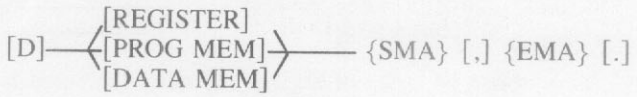
BREAKPOINT



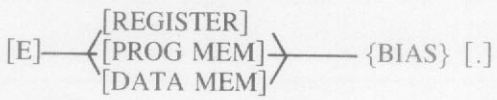
CLEAR



DUMP



ENTER



EPROM PROGRAM FOR DEBUG (8748) [7] {SMA} [,] {EMA} [,] {SEP} [.]

EPROM PROGRAM NO DEBUG (8748,41,55) [3] {SMA} [,] {EMA} [,] {SEP} [.]

EPROM COMPARE [8] {SMA} [,] {EMA} [,] {SEP} [.]

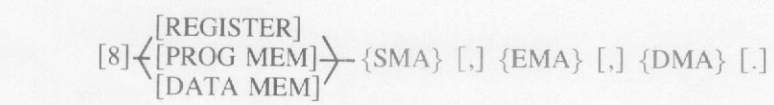
FETCH EPROM [F] {SMA} [,] {EMA} [,] {SEP} [.]

HEX CALCULATOR [6] {DATA} [,] {DATA} [.]

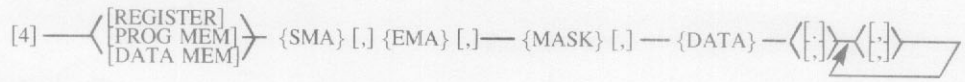
MAP P2 [2] {DIR} [.]

Each bit of DIR is direction, 1=input, 0=output.

MOVE MEMORY



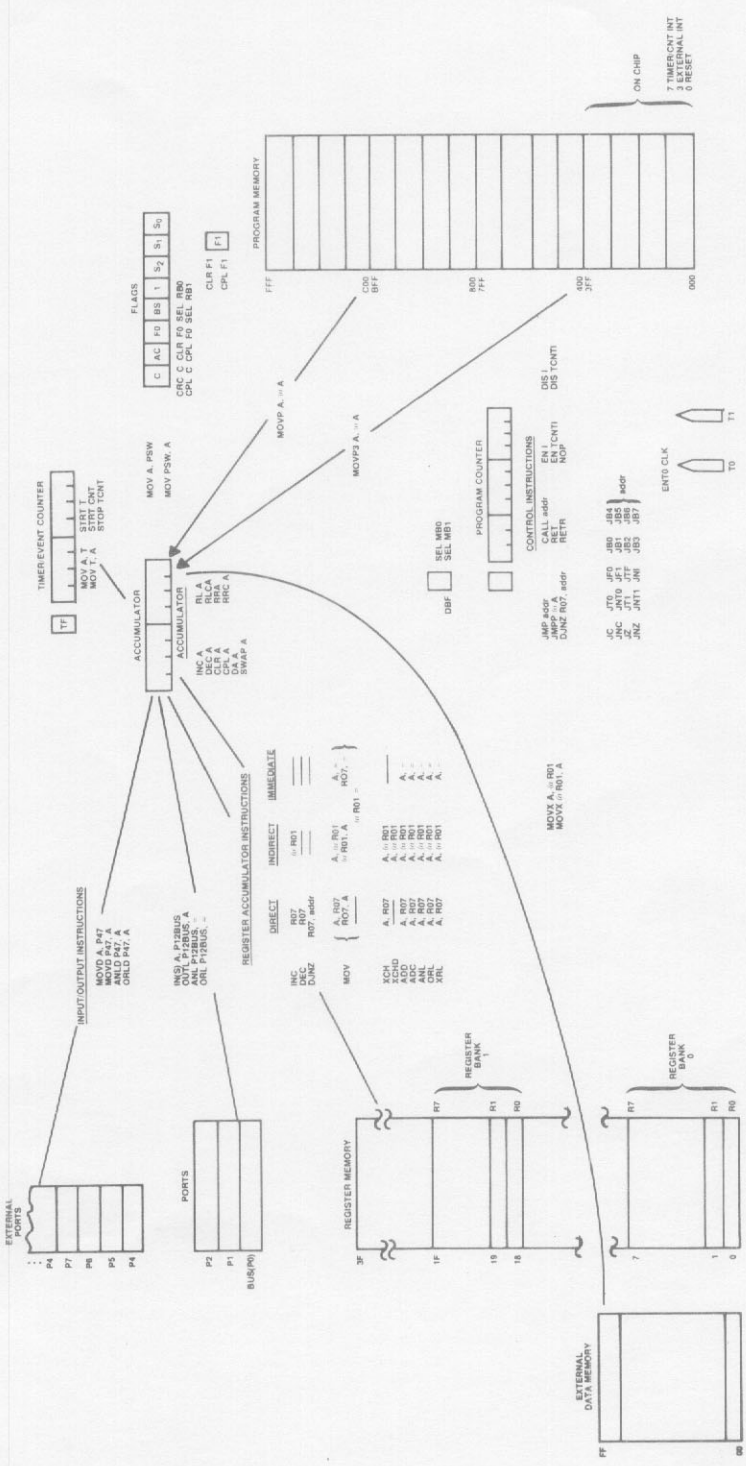
SEARCH BYTE



SEARCH 2 BYTES

{BIAS}	Bias offset to load address
{BKA}	Breakpoint address
{DATA}	Data
{DIR}	Direction for lines of Port 2
{DMA}	Destination memory address
{EMA}	Ending memory address
{HDATA}	High byte of data
{HMASK}	High byte of mask
{LDATA}	Low byte of data
{LMASK}	Low byte of mask
{MASK}	Mask
{SEP}	Starting EPROM address
{SMA}	Starting memory address

APPENDIX F
MICROMAP



APPENDIX G
INSTRUCTION SET SUMMARY

BY MNEMONIC

• ADD A, R0	68
R1	69
R2	6A
R3	6B
R4	6C
R5	6D
R6	6E
R7	6F
• ADD A, @R0	60
@R1	61
• ADD A, #data	03 □□
• ADDC A, R0	78
R1	79
R2	7A
R3	7B
R4	7C
R5	7D
R6	7E
R7	7F
• ADDC A, @R0	70
@R1	71
• ADDC A, #data	13 □□
ANL A, R0	58
R1	59
R2	5A
R3	5B
R4	5C
R5	5D
R6	5E
R7	5F
ANL A, @R0	50
ANL A, @R1	51
ANL A, #data	53 □□
ANL BUS, #data	98 □□
P1, #data	99 □□
P2, #data	9A □□
CALL 0addr	14 □□
1addr	34 □□
2addr	54 □□
3addr	74 □□
4addr	94 □□
5addr	B4 □□
6addr	D4 □□
7addr	F4 □□
CLR A	27
• CLR C	97
CLR F0	85
CLR F1	A5
CPL A	37
• CPL C	A7
CPL F0	95
CPL F1	B5
• DA A	57
DEC A	07
DEC R0	C8
R1	C9
R2	CA
R3	CB
R4	CC
DEC R5	CD
R6	CE
R7	CF
DIS I	15
DIS TCNTI	35
DJNZ R0,addr	E8 □□
R1,addr	E9 □□
R2,addr	EA □□
R3,addr	EB □□
R4,addr	EC □□
R5,addr	ED □□
R6,addr	EE □□
R7,addr	EF □□
EN I	05
EN TCNTI	25
ENT0 CLK	75
INS A, BUS	08
IN A, P1	09
IN A, P2	0A
INC A	17
INC R0	18
INC R1	19
INC R2	1A
INC R3	1B
INC R4	1C
INC R5	1D
INC R6	1E
INC R7	1F
INC @R0	10
INC @R1	11
JB0 addr	12 □□
JB1 addr	32 □□
JB2 addr	52 □□
JB3 addr	72 □□
JB4 addr	94 □□
JB5 addr	B2 □□
JB6 addr	D2 □□
JB7 addr	F2 □□
JC addr	F6 □□
JF0 addr	B6 □□
JF1 addr	□□
JMP 0addr	04 □□
1addr	24 □□
2addr	44 □□
3addr	64 □□
4addr	84 □□
5addr	A4 □□
6addr	C4 □□
7addr	E4 □□
JMPP @A	B3 □□
JNC addr	E6 □□
JNI addr	86 □□
JNT0 addr	26 □□
JNT1 addr	46 □□
JNZ addr	96 □□
JTF addr	16 □□
JT0 addr	36 □□
JT1 addr	56 □□
JZ addr	C6 □□
MOV A, #data	23 □□
MOV A, PSW	C7
MOV A, R0	F8
R1	F9
R2	FA
R3	FB
R4	FC
R5	FD
R6	FE
R7	FF
MOV A, @R0	F0
@R1	F1
MOV A, T	42
• MOV PSW, A	D7
MOV R0, A	A8
R1, A	A9
R2, A	AA
R3, A	AB
R4, A	AC
R5, A	AD
R6, A	AE
R7, A	AF
MOV R0, #data	B8 □□
R1, #data	B9 □□
R2, #data	BA □□
R3, #data	BB □□
R4, #data	BC □□
R5, #data	BD □□
R6, #data	BE □□
R7, #data	BF □□
MOV @R0, A	A0
MOV @R1, A	A1
MOV @R0, #data	B0 □□
@R1, #data	□□
MOV T, A	62
MOVD A, P4	0C
P5	0D
P6	0E
P7	0E
MOVD P4, A	3C
P5, A	3D
P6, A	3E
P7, A	3F
MOVP A, @A	A3
MOVP3 A, @A	E3
MOVX A, @R0	80
@R1	81
MOVX @R0, A	90
@R1, A	91
NOP	00
ORL A, R0	48
R1	49
R2	4A
R3	4B
R4	4C
R5	4D
R6	4E
R7	4F
ORL BUS, #data	88 □□
P1, #data	89 □□
P2, #data	8A □□
ORLD P4, A	8C
P5, A	8D
P6, A	8E
ORLD P7, A	8F
OUTL BUS, A	02
P1, A	39
P2, A	3A
RET	83
RETR	93
RL A	E7
• RLC A	F7
RR A	77
• RRC A	67
SEL MB0	E5
SEL MB1	F5
SEL RB0	C5
SEL RB1	D5
STOP TCNT	65
STRT CNT	45
STRT T	55
SWAP A	47
XCH A, R0	28
R1	29
R2	2A
R3	2B
R4	2C
R5	2D
R6	2E
R7	2F
XCH A, @R0	20
XCH A, @R1	21
XCHD A, @R0	30
@R1	31
XRL A, R0	D8
R1	D9
R2	DA
R3	DB
R4	DC
R5	DD
R6	DE
R7	DF
XRL A, @R0	D0
@R1	D1

• CARRY FLAG AFFECTED

All mnemonics copyright © 1976, 1977, 1978 Intel Corporation

APPENDIX H
HEXADECIMAL/BINARY CONVERSION TABLE

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

HEXADECIMAL COLUMNS												
6		5		4		3		2		1		
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	
0	0	0	0	0	0	0	0	0	0	0	0	
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1	
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2	
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3	
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4	
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5	
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6	
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7	
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8	
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9	
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10	
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11	
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12	
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13	
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14	
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15	
0123		4567		0123		4567		0123		4567		
BYTE				BYTE				BYTE				

ASCII CHARACTER SET (7-BIT CODE)

LSD \ MSD		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

POWERS OF 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$

POWERS OF 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

APPENDIX I
ACCESS CODE/LSN P2 MAP SUMMARY

Access Code	Program Memory	System I/O & Calls	Expansion Memory & I/O	OUTL Port 0	Allowed LSN P2 Map
0	RAM	No	No	Yes	output (0) only
1	RAM	No	Yes	No	input or output
2	RAM	Yes	No	No	output only
3	On-chip ROM/EPROM	No	No	Yes	input or output
4	On-chip ROM/EPROM	No	Yes	No	input or output
5	On-chip ROM/EPROM	Yes	No	No	output only

APPENDIX J
EXPANDED ACCESS CODES WITH 6 MHZ OPTION

For those systems equipped with the 6 MHz upgrade option, the following twelve access codes are operative:

	Access =	OUTL Port 0	Expansion Memory	Expansion I/O	System I/O & Calls
Prompt RAM for Program Memory	0	Yes	No	No	No
	1	No	Yes	No	No
	2	Yes	No	No	Yes
	3	Yes	No	Yes	No
	4	No	Yes	Yes	No
	5	Yes	No	Yes	Yes
On-chip ROM/EPROM for Program Memory	10	Yes	No	No	No
	11	No	Yes	No	No
	12	Yes	No	No	Yes
	13	Yes	No	Yes	No
	14	No	Yes	Yes	No
	15	Yes	No	Yes	Yes

The equivalents of the original 6 access codes are:

Old Access	New Access
0	0
1	4
2	2 (with OUTL, too)
3	10
4	14
5	12 (with OUTL, too)

INDEX

REQUEST FOR READER'S COMMENTS

The Microcomputer Division Technical Publications Department attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____
TITLE _____
COMPANY NAME/DEPARTMENT _____
ADDRESS _____
CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

First Class
Permit No. 1040
Santa Clara, CA

BUSINESS REPLY MAIL

No Postage Stamp Necessary if Mailed in U.S.A.

Postage will be paid by:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Attention: MCD Technical Publications