MPF-I/88

User's Manual

MPF-I/88

User's Manual

Table of Contents

Chapter 5 Monitor Commands

Chapter 6 Line Assembler

Chapter 7 Useful Subroutines

Appendices

Chapter 1

Getting to Know Your MPF-I/88

1.1 Unpacking

After unpacking, check whether your MPF-I/88 is complete with the following:

* The MPF-I/88 itself
* 1 power supply adapter
* 3 manuals
* 2 close jumpers

1.2 What is on the MPF-I/88 Main Board?

Before we go any further, the user is strongly suggested to spend a few minutes in order to get acquainted with the various parts of the MPF-I/88 by looking at Fig. 1-1 below:

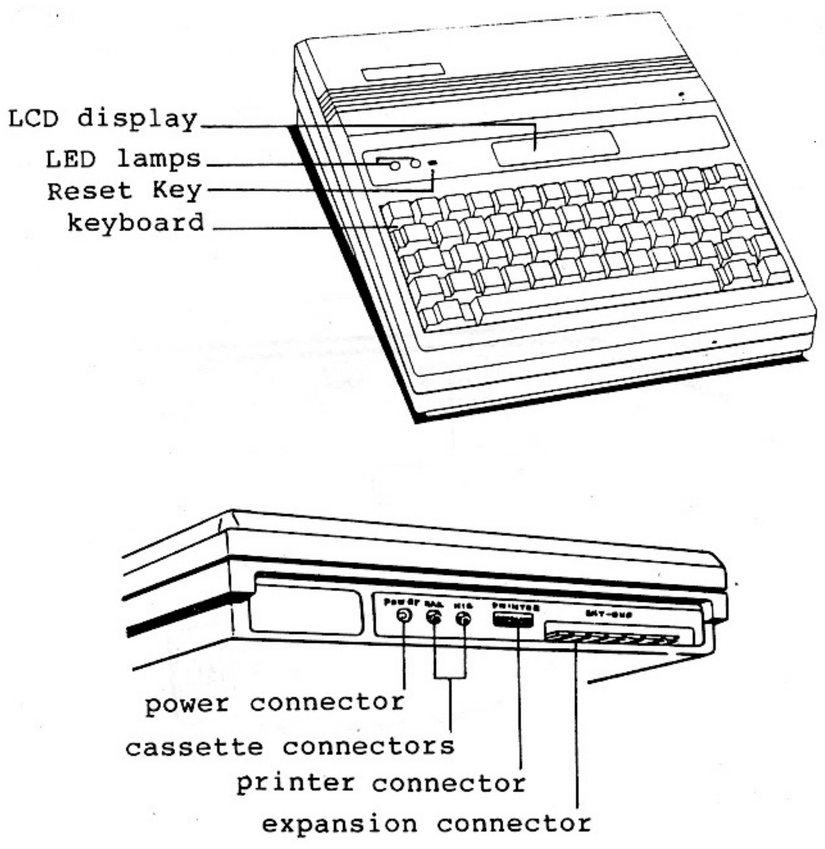


Fig. 1-1 MPF-I/88 Physical Layout

1.3 What can the MPF-I/88 Do?

The MPF-I/88 is a versatile machine and can perform lots of things like:

1. Execute programs, play games, let you practice on assembly language,etc.
2. Convert assembly language into machine code line by line as you type it.
3. Check and debug errors found during program execution.
4. Load data from and store data to tape.
5. Examine system status by checking the contents of registers and memory.
6. Allow you to communicate with the system by using a built-in monitor program.

1.4 How to Set up the MPF-I/88 for Use?

It is very easy to use the MPF-I/88, but first you have to make the proper connections by doing the following steps:

-Connect the power adapter to the power source

-Insert the plug from the power adapter into the power adapter jack as shown in Fig. 1-2

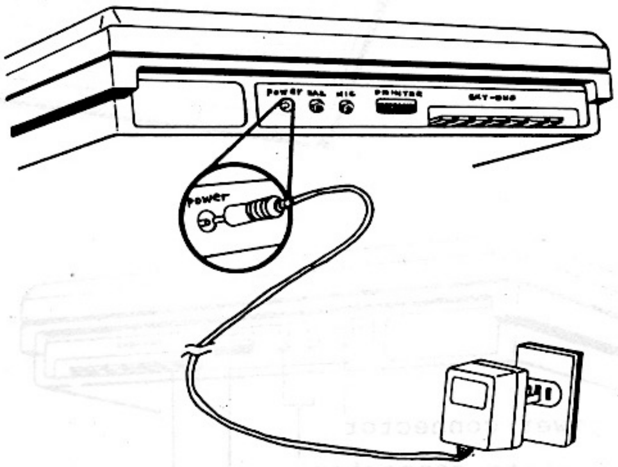


Fig. 1-2

If you have set up the MPF-I/88 properly, the display will show:

```
          MPF-I/88
   MONITOR VERSION   1.Ø
```

Fig. 1-3

then,

```
          MPF-I/88
   (c)  1984  MULTITECH
```

Fig. 1-4

then,

```
          MPF-I/88

   CHECK RAM   ■
```

Fig. 1-5

then,

```
          MPF-I/88

   CHECK ROM    ■
```

Fig. 1-6

and finally,

```
  > ■
```

Fig. 1-7

At this point, the computer is ready to receive instuctions from you. Details on the various commands you can use on the MPF-I/88 will be discussed in following chapters.

1.5 What to Do if the MPF-I/88 Didn't Respond as It should?

If after plugging the MPF-I/88, the display doesn't show anything, first check if the proper connections had been made, like if the power adapter is connected to the power source or if the power source is turned on or not. Then, if it still doesn't respond, contact your local Multitech dealer.

Sometimes, the MPF-I/88 might display the following messages:

1.

```
              MPF-I/88

   RAM      CHECK ERROR
```

Fig. 1-8

Note: This means that there is error in the RAM.

2.

```
              MPF-I/88

ROM CHECK ERROR
```

Fig. 1-9

If you get this error message, that means the ROM consists of one 27128 chip and it has an error.

Chapter 2

Introduction to System Hardware

Now you are familiar with the basic configuration of the MPF-I/88. Before you go step to the next step, let us introduce its hardware specifications in more details.

2.1 The Three Main Parts of MPF-I/88

The MPF-I/88 is composed of the CPU (central processor unit), memory and other devices. Data transmission between the CPU and memory or between the CPU and devices is through the system bus.

2.2 What Is the CPU?

The CPU is the heart of the microcomputer. The CPU controls all operations within the system unit. The MPF-I/88 uses the Intel's 8Ø88 16-bit microprocessor as its CPU. It operates at a clock rate of 4.77 MHz. This is a measure of the speed at which the CPU executes its instructions. The cycle time is Ø.21 microseconds.

2.3 What Is Memory?

Memory includes ROM, RAM, and an user expansion area. The overall address range is from ØØØØØH to FFFFFH. There are the lowest and highest number that the 8Ø88 microprocessor can use as memory address. However, your MPF-I/88 only uses a small subset of this potential range of memory.

ROM stands for "read only memory" and refers to a special kind of memory chip which retains its contents even when the power is turned off. ROM memory chips are typically used to store software that can be used right away when the computer is turned on. On the MPF-I/88, the ROM memory is used to store the monitor and line assembler software you will learn about below.

The MPF-I/88 has three sockets for ROM chips. Each of these sockets can contain either an 8K byte ROM chip or a 16K byte ROM chip. The standard MPF-I/88 comes with one 16K byte ROM chip. The ROM memory can be expanded to a total of 48K bytes by using three 16K ROM chips.

The address ranges for the three ROM sockets are as follows.

socket name	address range
ROMØ:	FCØØØH - FFFFFH
ROM1:	F8ØØØH - FBFFFH
ROM2:	F4ØØØH - F7FFFH

RAM stands for "random access memory". While it is not clearly implied by the names, the difference between ROM and RAM is that RAM can be written to as well as read and its contents disappear when the power is turned off. RAM is typically used in microcomputers as a working area for programs and data. The larger this RAM memory area is, the longer and more complicated programs can be.

The MPF-I/88 also has three sockets for RAM chips. Each socket can contain either a 2K byte RAM chip or an 8K byte RAM chip. The standard MPF-I/88 comes with two 2K byte RAM chips for a total user memory of 4K. The RAM directly on the system board can be expanded to 24K by using three 8K RAM chips. The two kinds of RAM chips cannot be used together at the same time. Please see the Appendix for a discussion of special considerations for installing 8K RAM chips.

The address ranges for the three RAM sockets are as follows:

socket name	address range when using 2K byte RAM
RAMØ:	ØØØØØH - ØØ7FFH
RAM1:	ØØ8ØØH - ØØFFFH
RAM2:	Ø1ØØØH - Ø17FFH

socket name	address range when using 8K byte RAM
RAMØ:	ØØØØØH - Ø1FFFH
RAM1:	Ø2ØØØH - Ø3FFFH
RAM2:	Ø4ØØØH - Ø5FFFH

(Note: The underlined address ranges show the ranges for the standard configuration of the MPF-I/88.)

2.4 What Input/Output Devices Does the MPF-I/88 Include?

The devices that the system includes are:

Buzzer: The buzzer can be controlled by the user to make tones of different pitches. It also generates sounds when storing data onto the tape.

Display: The display of the machine is a dot matrix LCD (liquid crystal display). Its physical format is 2Ø characters by 2 lines. However, the system software supports a 24-line logical display. The display can show any two-line section of the 24-line logical display at one time. You can use the key combination ALT-A and ALT-Z to scroll up and down in the 24-line logical display.

Keyboard: The keyboard has a total of 59 keys, including alphanumeric keys (from A to Z, from Ø to 9) and function keys. The keyboard uses rubber type contact technology.

Printer Port: The MPF-I/88 has a 16-pin Centronics printer port.

Audio Tape Interface: The machine can be connected to a monaural cassette tape recorder to store data.

LED Lamp: There are two LEDs (light emitting diode) to the upper left of the keyboard. One is green and the other is red. The function of the green LED lamp is similar to the buzzer. It always illuminates when you load data into the tape. The red LED lamp illuminates when the CPU executes the HLT instruction or the system is in the HOLD state.

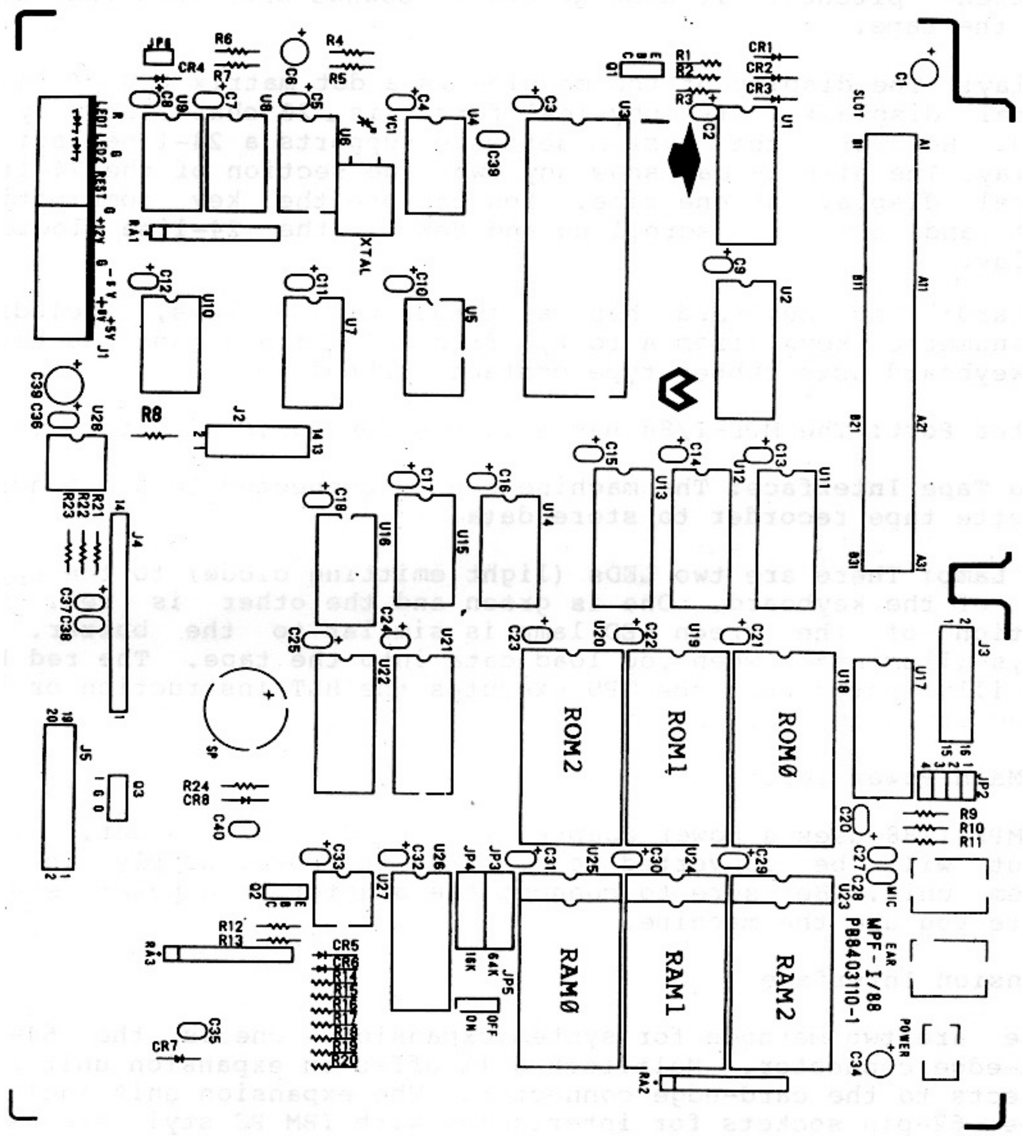
The Main Power Input

The MPF-I/88 uses a power adapter with a DC 9V 1A output. This output will be converted to DC 5V by the power supply in the system unit. Be sure to connect the adapter to a power source before you use the machine.

Expansion Interface

There are two methods for system expansion. One is the 64-pin card-edge connector. Multitech will offer an expansion unit that connects to the card-edge connector. The expansion unit includes three 62-pin sockets for interfacing with IBM PC style expansion boards and a 4Ø-pin connector for interfacing with peripherals developed for use with other members of the MPF-IP family.

In addition, we have reserved a place next to the edge connector for a 62-pin H connector. You can solder a slot on the reserved place, and then plug an expansion board into the slot.



MPF-I/88
PB8403110-1
ROM0
ROM1
ROM2
RAM0
RAM1
RAM2
SLOT
XTAL
POWER
EAR
MIC
SP
JP1
JP2
JP3
JP4
JP5
J1
J2
J3
J4
J5
RA1
RA2
RA3
16K
64K
ON
OFF
LED1
LED2
RESET
+12V
-5V
+5V

Chapter 3

Entering and Executing a Sample Program

3.1 A Sample Program

We will assume, when referring to the following example in this chapter, that you are sitting at your MPF-I/88, have turned on the system, and have a basic familiarity with 8Ø88 assembly language.

Here is our first assembly program. It calculates the sum of all the integers between 1 and 1ØØ and then prints the total on the screen. Type it into the machine as a first example of using the MPF-I/88 to see what happens during typing according to the following statements once a prompt '>' is shown on the screen. The below characters with boldface type should be typed in by you, others are prompted by the system.

```
        >■

        >A ØØ8Ø:Ø ↵

        ØØ8Ø:ØØØØ MOV  AX,Ø ↵

        ØØ8Ø:ØØØ3 MOV  CX,64 ↵

        ØØ8Ø:ØØØ6 ADD  AX,CX ↵

        ØØ8Ø:ØØØ8 LOOP ØØØ6 ↵

        ØØ8Ø:ØØØA INT  1Ø ↵

        ØØ8Ø:ØØØC INT  7 ↵

        ØØ8Ø:ØØØE ↵

        >■
```

After you have typed in the command and the cariage return key, the LCD display will show:

```
e
>
```

But that is not the sum of all the intergers from 1 to 1ØØ. Now press ALT–A (Press the A key while holding down the ALT key). What will you see on the LCD display? Does it look like the picture below:

```
13BAProgram terminat
e
```

This message tells you that the program has been terminated and the result of the program, which is 13BA. However, you get the answer of 5Ø5Ø by adding all the integers ranging from 1 to 1ØØ. Why does MPF-I/88 return the value 13BA as the answer?

This number is in hexadecimal, or base 16. It is equal to 5Ø5Ø in decimal, or base 1Ø.

Note that the answer will be scrolled off the top of the screen once the program is executed, whereas it doesn't mean that the answer has been lost. You can press the ALT key while holding down the A key to scroll up the screen. Please refer to section 5.2.2, the Display Buffer, for more details.

Now let us look at it more closely. The first line of our example:

```
>■
```

This line will be always displayed on the screen by the system each time you power up the machine or after you have executed a program. It means that the system is waiting for you to enter a monitor command. The ">" symbol is referred to as the monitor prompt character. The symbol "■" is called the cursor, and is always displayed on the screen to show your current typing position.

The second line of the example is:

```
>A ØØ8Ø:Ø ↵
```

This line is an example of a monitor command. The symbol "A" tells the system to enter the line assembler; and the number "ØØ8Ø:Ø" tells the system where your program is to begin (i.e., the starting address of the program to be entered). The symbol ↵ represents the carriage return key. It must be entered to terminate a command and tell the system to execute the command you have just typed. Note that the blank between "A" and "ØØ8Ø:Ø" is optional.

Each time you enter an instruction and press the carriage return key, the line assembler immediately translates the instruction into machine code and prompts you for the next line with a number that will be the address of the next instruction. You can leave the line assembler and return to the monitor prompt ">" .by pressing return without typing any instruction first.

Now let us look at the first line of the main body of the program:

```
        0080:0000 MOV  AX,0
```

The number "0080:0000" is typed on the screen by the system after you enter the command "A 0080:0" as described above. The instruction "MOV AX,0" following the number "0080:0000" means to move the 16-bit quantity 0 (HEX) into the AX register which is sometimes called the accumulator.

Because this instruction will occupy 3 bytes of memory, the system will prompt with "0080:0003" in the next line after you press the carriage return key.

The second line of the program is:

```
        0080:0003 MOV  CX,64
```

This instruction is used to move the 16-bit quantity 64 into the CX register referred to as the count register. All numbers used in the line assembler are in hexadecimal; thus 64 hex = 100 decimal. The CX register is typically used to control the number of iterations a loop will perform and will be also decremented by loop operations. Because we want to calculate the sum of all the integers between 1 and 100, we need to loop 100 times.

The third line of the program is:

```
        0080:0006 ADD  AX,CX
```

This instruction is used to add the contents of the CX register to the contents of the AX register, and store the result into the AX register.

The fourth line of the program is:

```
        0080:0008 LOOP  0006
```

This instruction decrements the CX register's contents by one each time the instruction is executed. Then, the microprocessor will check if the CX register has been decremented to 0. In our example here, if the content of the CX register is greater than 0, control will be transferred back to the ADD instruction at the address specified by the operand 0006 in the LOOP instruction.

Otherwise, the next instruction "INT 10" is executed. Let us go ahead with the fifth line of our program:

```
        0080:000C INT  10
```

The mnemonic INT shown above is called an interrupt. In our program, the instruction INT 1Ø means that we are going to output our answer (13BA) onto the screen via a service program which is stored in ROM and the function of which is to write the contents of the AX register onto the screen.

The sixth line of our program is:

```
ØØ8Ø:ØØØE INT  7
```

The instruction means that control of the system is tranferred from the program to the Monitor. All program should end with this instruction.

The last line of our program is:

```
ØØ8Ø:ØØØE ↵
```

Since you have already typed in the last line of the program, at this point you should simply press ↵ in order to return to the monitor prompt.

All the contents of the RAM memory, such as the program you just typed in, will disappear when you turn off the MPF-I/88. Therefore, if you would like to keep the program for future use, you can store it onto tape. The next chapter will introduce you what kinds of commands you can enter for storing data in memory onto tape and for loading the data on tape back into memory.

Chapter 4

Using a Tape Recorder with MPF-I/88

4.1 Recording on Tape

We assume that you have gone through the previous chapters on the system configuration, and know where the phone jacks for the tape recorder reading and writing are.

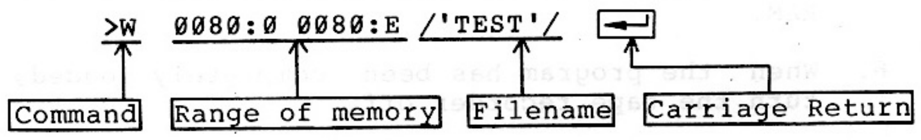
The following procedure will show you how to record your program on tape:

1. First, plug one end of the recorder cable to the MIC (microphone) jack on the system board and the other end to the MIC jack of the tape recorder.

2. Place a cassette in the tape recorder, rewind it, and then play it until you are past the leader. If your recorder has a counter, allow it at least five counts.

3. Put the tape recorder in the record mode, and set the voice volume control switch to a medium position.

4. Then, enter the command W (for write) following the prompt >. Refer to the following example:



```
>W  ØØ8Ø:Ø ØØ8Ø:E /'TEST'/
```

   Note:
   The W command is described in detail in section 5.3.17. After the carriage return key is pressed, the machine will write data from the RAM memory to the cassette. The file is stored on tape with the file name 'TEST'

5. You will hear the sounds used to encode your program on tape coming out of the speaker.

6. When recording has been completed, stop the recorder.

4.2 Loading from Tape

To load a program from tape, follow the following procedure:

1. First, plug one end of the recorder cable to the EAR (earphone) jack on the system board and the other end to the EAR jack of the tape recorder.

2. Place the cassette storing the file which you are going to use in the tape recorder and rewind it to the beginning.

3. Enter the command R (for read) with the file name following the prompt >. Refer to the following:

```
          >R  /'TEST'/    [↵]
           ↑       ↑       ↑
      [Command] [Filename] [Carriage Return]
```

   Note:

   The R command is described in details in section 5.3.18.

4. Place the tape recorder in the Read Mode by depressing PLAY.

5. Then, press the [↵] key. The machine will now begin reading data from the tape to its RAM.

6. When the program has been completely loaded, turn the tape recorder off.

Chapter 5

Monitor Commands

The monitor is a very powerful and easy to use program that allows you to communicate interactively with the MPF-I/88. You have already been shown examples of the use of some of the monitor commands, such as A and G. This chapter gives you a detailed explanation of the use of each of the monitor commands.

5.1 Functions of the Monitor Program

1. Directly assemble 8Ø88 instructions from mnemonic form to machine code by line.

2. Automatcally detect the bugs of the program written by you.

3. Set up to 3 breakpoint addresses at which the program will stop temporarily when executed.

4. Display and change memory contents.

5. Display and change register contents.

6. Find a certain string of charaters.

7. Delete a block of data in memory.

8. Insert a block of data in memory.

9. Load and store memory to and from a cassette tape recorder.

1Ø. Allow a CRT or other kind of terminal to be used as the control console (including the keyboard and the monitor) of the MPF-I/88.

5.2 Syntax Notation

The following syntax notation is used throughout chapter 5 in descriptions of the monitor command systax:

[ ]	Square brackets indicate that the enclosed entry is optional.
< >	Angle brackets indicate data or address you must enter.
....	Ellipses indicate that an entry may be repeated as many times as needed or desired.

All other punctuation, such as colons, slash marks, and apostrophe signs, must be entered exactly as shown.

5.2.1 Specifying address when using the monitor

In the following descriptions of monitor command usage, whenever the parameter "addr" appears, it stands for address, and can be entered in either one of the following two forms:

Offset or **Seg:Offset**

e.g. **Ø** or **ØØ8Ø:Ø**

The abbreviation "Seg" stands for segment. The 8Ø88 microprocessor features a segmented memory architecture in which effective addresses are formed by adding a segment address to an offset address within the segment according to the diagram below:

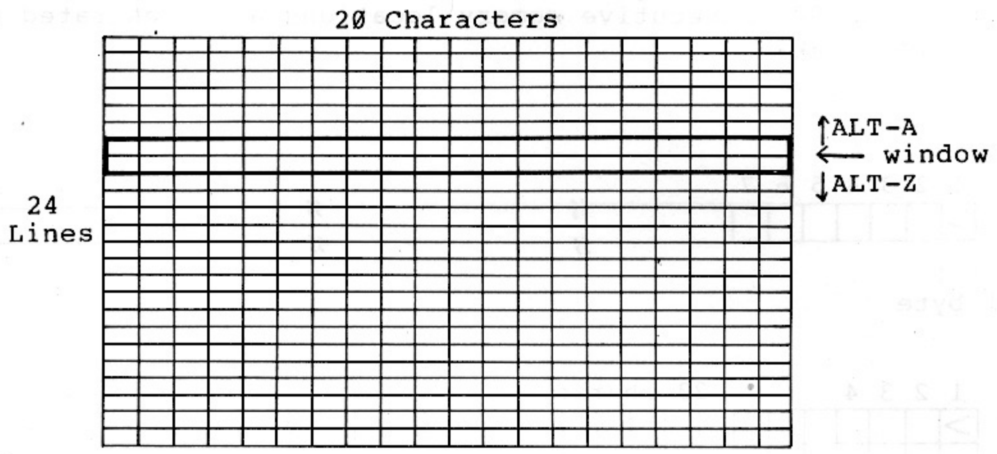
```
   Ø Ø 8 Ø Ø    Segment Address
 +   Ø Ø 1 B    Offset  Address
 -----------
   Ø Ø 8 1 B    Effective Address
```

If when entering an address while using the monitor you specify only the offset address, the monitor will use the current default segment address. You can use the abbreviations CS:, DS:, SS:, and ES: as the segment address in an address parameter to instruct the monitor to use the current contents of one of these registers as the segment value. You will probably find that unless you write very large programs with the MPF-I/88, you will find it most convenient to let the CS register point to the beginning of your program and deal only with offset address values when using the monitor. By the way, you can also use the segment override mnemonics to replace the segment value for convenience, such as CS, DS, ES, and SS.

When you turn on the MPF-I/88 and the monitor is initialized, the default segment value will be ØØ8Ø. It is recommended that when using the monitor and line assembler, you use segment values greater than ØØ8Ø. Otherwise, you may overwrite data areas used by the monitor.

5.2.2 The Display Buffer

The display buffer is an area in the system memory. It is stored with the information to be displayed on the logical screen. Since the logical screen of MPF-I/88 is 2Ø characters by 24 lines, you can visualize the display buffer as a block of memory illustrated as follows:



The physical display (2Ø characters by 2 lines) can be visualized as a window (or the view finder of a camera) that allows you see any part of the logical screen. You can move the window upward to see the upper portion of the logical screen or move the window downward to see the lower portion of the logical screen.

ALT-A

When you move the window upward, you are actually scrolling down the logical screen. The window can be moved upward by pressing ALT-A.

ALT-Z

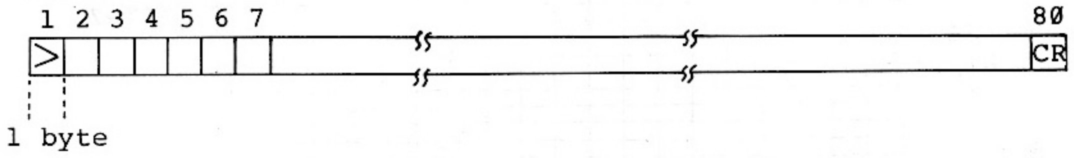
When you move the window downward, you are actually scrolling up the logical screen. The window can be moved downward by pressing ALT-Z.

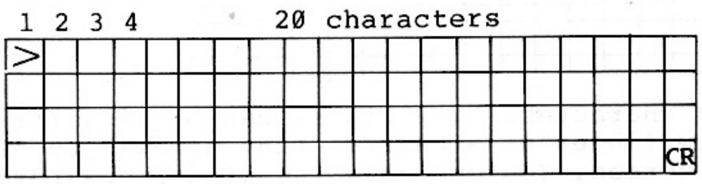
You can always use ALT-A and ALT-Z to move the window to a desired location to see a desired portion of the logical screen. To return to the location where the cursor was previously located, you can type any key on the keyboard.

5.2.3 Editing a Command Line

The Command Line Buffer

When you enter a command line, the command line is actually stored in an area in the system memory. That area is generally referred to as a command line buffer. The length of the command line buffer is 8Ø bytes. You can visualize the command line buffer as 8Ø consecutive memory locations as illustrated in the diagram below:





The first byte of the command line buffer is used to stored the monitor prompt character ">". The last memory location of the command buffer only accepts the ASCII code for the carriage return key. Thus, the command buffer can accept a maximum of 78 characters (or bytes of information).

Editing Functions

MPF-I/88 monitor program provides editing functions so that when you made a typing error inadvertently in a long command line you don't have to re-type a whole command line.

The editing functions are supported using four special function keys: F1, F2, SHIFT-F1, and SHIFT-F2.

The following is a brief description of these function keys:

F1:	Copy one character.
F2:	Copy all the remaining characters in a command line.
SHIFT-F1:	Insert character.
SHIFT-F 2:	Delete character.

As you get more and more familiarized with the monitor commands, you may quite often find yourself entering the same command lines or a command line that is similiar to a previously entered command line.

In this case, you can use F1 and F2 to copy a character or the characters which you have already typed into the command line buffer.

We will use some examples to make it easier for you to understand the use of these special editing function keys. But before showing the examples, it is worthwhile to see how the monitor handles these function keys.

Each time a key is pressed, the monitor will determine if it is one of F1, F2, SHIFT-F1, and SHIFT-F2, if one of these keys are entered, an editing function is invoked. Each time an editing function is invoked, the monitor will copy the contents of the command line buffer into a temporary working buffer so that contents of the temporary working buffer can be edited.

The monitor maintains an internal pointer which always point to the location where the cursor is currently located. In other words, the position of the internal pointer corresponds to the position of the cursor. Any time a change is to be made in the temporary working buffer, the cursor should be moved to the location where the change is to be made. As soon as a change is made in the temporary working buffer, the new contents of the temporary working buffer is sent back the command line buffer so that the new command line can be executed.

Example 1 -- F2

1. Suppose you want to examine the contents of the memory range starting from memory location Ø through 5. You can type in the command line as follows:

   ```
   >M Ø:Ø 5
   ```

   After the carriage return is pressed, the contents of the specified memory range will be displayed.

2. F2

   At this time, if you want to examine the contents of the same memory range again, you can simply type the function key F2. After F2 is pressed, the screen will automatically display:

   ```
   >M Ø:Ø 5
   ```

   At this point, you can press the carriage return key to have the screen display the contents of the specified memory range.

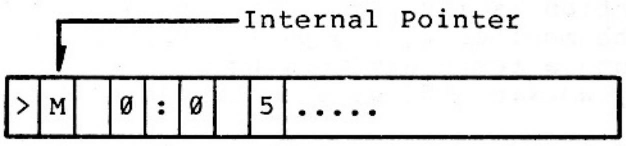
If you want to try function key F1, continue with the next example. But at this time don't press any key on the keyboard until you are told to.

Example 2 -- F1

If you want to examine the contents of memory location Ø through 1Ø, you have two ways to save time. One of them is to use the F1 key.

Copy one character at a time

1. Once F1 is pressed, the contents of the current command line buffer (M Ø:Ø 5) are copied to the temporary working buffer, and the internal pointer is moved as illustrated:



*** Temporary Working Buffer ***

2. At this time you can press F1 to copy the character M to the command line buffer. Pressing F1 repeatedly will copy the entire command line M Ø:Ø 5 into the command line buffer. Since we want to enter the command M Ø:Ø 1Ø, you can F1 until the screen shows:

```
>M Ø:Ø █
       |
       └──► This is where the cursor is positioned.
```

   At this point you can type 1 and Ø on your keyboard so that the display becomes:

```
>M Ø:Ø 1Ø
```

3. Pressing the carriage return key at this time will cause the contents of memory locations Ø through 1Ø to be displayed.

Copy the whole command line

A second way to enter the command line M Ø:Ø 1Ø is to use the function key F2 to copy the whole previously entered command line into the temporary working buffer, update the contents of the temporary working buffer, and then copy the updated command line from the temporary working buffer to the command line buffer.

1. Suppose the previously entered the command line is M Ø:Ø 5. You can type F2 to copy the whole previously entered command line into the temporary working buffer. Once F2 is pressed, the screen display will show:

```
>M Ø:Ø 5
```

2. You can press the backspace key to delete 5 in the command line. After pressing the backspace key, the display will become:

   ```
   >M Ø:Ø ■
   ```

3. Now you can press 1 and Ø to update the command line.

4. After pressing the carriage return key, the contents of memory locations Ø through 1Ø will be displayed.

   If you want to try function key SHIFT-F1, continue with the next example. But at this time don't press any key on the keyboard until you are told to.

Example 3 -- SHIFT-F1

SHIFT-F1 (hereafter referred to as S-F1) is used to insert characters in a command line.

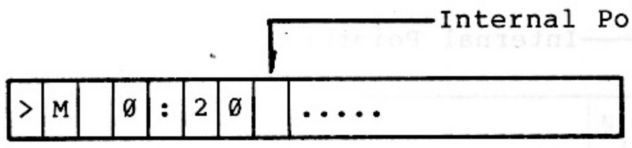
Suppose the current command line buffer is stored with the command line M Ø:Ø 1Ø, and you want to examine the contents of memory locations 2ØØH through 25ØH.

1. The first step is pressing F1 repreatedly until the display shows:

   ```
   >M Ø:■
   ```

2. Type S-F1 so that characters can be inserted into the command line.

3. Type 2 and Ø. At this time the temporary working buffer becomes:



*** Temporary Working Buffer ***

4. Type F1 until the screen becomes:

   ```
   >M Ø:2ØØ ■
   ```

5. Type 2, 5, and Ø. The LCD display will show:

   ```
   >M Ø:2ØØ 25Ø
   ```

At this time, the command line buffer is stored with this updated command line. After the carriage return key is pressed, the contents of memory locations 2ØØ through 25Ø will be displayed.

Example 4 -- SHIFT-F2

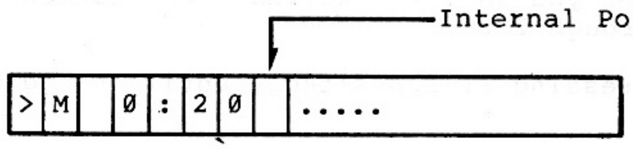
SHIFT-F2 (hereafter referred to as S-F2) is used to delete characters in a command line.

At this moment the current command line buffer is stored with the command line M Ø:2ØØ 25Ø, and you want to examine the contents of memory locations 2ØH through 5ØH.

1. The first step is pressing F1 repreatedly until the display shows:

   ```
   >M Ø:2Ø
   ```

2. Type S-F2 so that characters in the temporary working buffer (command line buffer) can be deleted. At this time the temporary working buffer becomes:

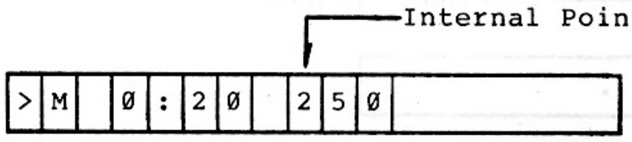


*** Temporary Working Buffer ***

3. Type F1. Now the display has changed to:

   ```
   >M Ø:2Ø █
   ```

   At this time the temporary working buffer becomes:



*** Temporary Working Buffer ***

4. Type S-F2 to delete 2.

5. Type F2 to copy the remaining characters in the temporary working buffer into the command line buffer. At this time, the display becomes:

   ```
   >M Ø:2Ø 5Ø
   ```

Pressing the carriage return key at this point will cause the contents of the memory range specified to be displayed.

5.2.4 Moving the Cursor in the Command Buffer

Suppose you want to make changes to some locations in the command line buffer that are already keyed in but not currently shown on the window (the physical display). What are you going to do?

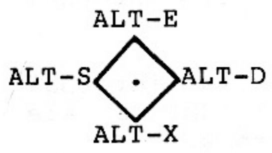
The MPF-I/88 monitor program allows you to move the cursor to a desired location in the command buffer and make changes using the special editing function keys as described previously.

To move the cursor, you can use the cursor movement keys. They are: ALT-E, ALT-S, ALT-D, and ALT-X.

To move the cursor up one line, type E while holding down the ALT key. ALT-X is used to bring down the cursor one line.

ALT-S is used to move the cursor left one position, while ALT-D is used to move the cursor right one position.

It seems that the functions of these cursor movement keys are difficult to remember. But look at the following chart (which is generally referred to as a cursor movement diamond) and then study where the keys E, S, D, and X are located on your keyboard.



Suppose the cursor is located in the center, you will find the E key is located exactly one line up the cursor and the X key is one line down the cursor. Now it is easy for you to remember the functions of the cursor movement keys by referring to their relative positions on the keyboard.

If you want to return the cursor to its position where it was located originally, simply type ALT-F (F for FAST).

5.2.5 Internal Pointers

The monitor program maintains several pairs of internal pointers to point to the memory locations which are currently accessed or to be accessed by some monitor commands. A pair of internal pointers are necessary to point to a specific memory location with one pointer pointing to the code or data segment and the other pointing to the offset address of that memory location.

An internal pointer itself is actually a specific memory location whose contents are set to a specific value by the monitor program during system cold reset but can be changed by entering a command parameter when you use a monitor command which allows you to alter the contents of the internal pointer.

Each internal pointer is identified with a label, which can be found in the monitor program source listing.

Take the monitor command A for example. It uses two internal pointers, A_TEMP_CS and A_TEMP_IP. Upon system power-up, the value of the internal pointer A_TEMP_CS is set to 8ØH, while the value of the internal pointer A_TEMP_IP is set to Ø. If you want to change the contents of these two pointers so that the two pointers will point to the memory location you desire, you can achieve that purpose by specifying the desired values as command parameters.

In fact, upon system power-up the contents of each pair of internal pointers is set the same way as the pair of internal pointers for the A command is initialized.

The L command uses I_TEMP_CS and I_TEMP_IP as its internal pointers. Upon system power-up, the contents of I_TEMP_CS are set to 8ØH, while the contents of I_TEMP_IP is set to Ø.

Some monitor commands uses common internal pointers. These commands are B, D, F, I, M, and T. They use the internal pointers -- TEMP_DS and TEMP_DP.

5.2.6 Monitor Command Listing

No.	Command	Command name	Page Location
1	A	ASSEMBLE	5-12
2	L	DISASSEMBLE	5-13
3	G	GO	5-17
4	S	STEP	5-18
5	B	BREAKPOINT	5-2Ø
6	C	CANCEL	5-22
7	X	REGISTER	5-23
8	M	MEMORY	5-24
9	I	INSERT	5-28
1Ø	D	DELETE	5-31
11	F	FIND	5-36
12	J	JUMP	5-38
13	T	TRANSFER	5-39
14	P	PAUSE	5-41
15	N	INPUT	5-41
16	O	OUTPUT	5-42
17	E	EXTEND	5-43
18	W	WRITE	5-44
19	R	READ	5-45

5.3 Explanation of the Monitor Commands

Part A — The Line Assembler and Disassembler

5.3.1 Command A — Enter the Line Assembler

Name: ASSEMBLER

Purpose: To let the user key 8Ø88 assembly language programs into memory and assemble them into machine code line by line.

Syntax:

(1) **A**
(2) **A <addr>**

Comments:

You can use this command to enter 8Ø88 assembly language program into the memory of the MPF-I/88. To leave the line assembler and return to the monitor prompt ">", press the carriage return key without typing an instruction.

addr: If there is no address parameter used with the A command, as shown in syntax one above, then the monitor will use the current contents of the CS register (which has a default value of ØØ8Ø) and the default offset value of Ø to determine the starting address of your program. By the way, the address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

Examples:

1. **>A**

```
    ØØ8Ø:ØØØØ  CLD
    ØØ8Ø:ØØØ1  MOV  SI,1Ø ;NUMBER 1Ø IS HEX
    ØØ8Ø:ØØØ4  LODSB
    ØØ8Ø:ØØØ5  CMP  AL,Ø
    ØØ8Ø:ØØØ7  JZ   D       ;D IS AN OFFSET, SEE "INT 7" BELOW
    ØØ8Ø:ØØØ9  INT  9       ;OUTPUT  A CHARACTER AT A TIME
    ØØ8Ø:ØØØB  JMP  4       ;4 IS AN OFFSET, SEE "LODSB" ABOVE
    ØØ8Ø:ØØØD  INT  7       ;RETURNING CONTROL TO MONITOR
    ØØ8Ø:ØØØF  NOP          ;NO OPERATION
    ØØ8Ø:ØØ1Ø  DB  'MICROPROFESSOR I/88'
    ØØ8Ø:ØØ23  DB   Ø
```

2. **>A 2Ø**

```
    ØØ8Ø:ØØ2Ø  CLD
    ØØ8Ø:ØØ21  MOV  SI,3Ø ;NUMBER 3Ø IS HEX
    ØØ8Ø:ØØ24  LODSB
    ØØ8Ø:ØØ25  CMP  AL,Ø
    ØØ8Ø:ØØ27  JZ   2D     ;2D IS AN OFFSET, SEE "INT 7" BELOW
    ØØ8Ø:ØØ29  INT  9      ;OUTPUT  A CHARACTER AT A TIME
    ØØ8Ø:ØØ2B  JMP  24     ;24 IS AN OFFSET, SEE "LODSB" ABOVE
    ØØ8Ø:ØØ2D  INT  7      ;RETURNING CONTROL TO MONITOR
    ØØ8Ø:ØØ2F  NOP         ;NO OPERATION
    ØØ8Ø:ØØ3Ø  DB  'MICROPROFESSOR I/88'
    ØØ8Ø:ØØ43  DB   Ø
```

3. **>A 9Ø:Ø**

or if the CS code segment register contains ØØ9Ø

>A CS:Ø

```
    ØØ9Ø:ØØØØ  CLD
    ØØ9Ø:ØØØ1  MOV  SI,1Ø ;NUMBER 1Ø IS HEX
    ØØ9Ø:ØØØ4  LODSB
    ØØ9Ø:ØØØ5  CMP  AL,Ø
    ØØ9Ø:ØØØ7  JZ   D      ;D IS AN OFFSET, SEE "INT 7" BELOW
    ØØ9Ø:ØØØ9  INT  9      ;OUTPUT  A CHARACTER AT A TIME
    ØØ9Ø:ØØØB  JMP  4      ;4 IS AN OFFSET, SEE "LODSB" ABOVE
    ØØ9Ø:ØØØD  INT  7      ;RETURNING CONTROL TO MONITOR
    ØØ9Ø:ØØØF  NOP         ;NO OPERATION
    ØØ9Ø:ØØ1Ø  DB  'MICROPROFESSOR I/88'
    ØØ9Ø:ØØ23  DB   Ø
```

5.3.2 Command L — Enter the Disassembler

Name: **DISASSEMBLER**

Purpose: To translate (disassemble) a block of machine codes in memory into 8Ø88 assembly instructions.

Syntax:

```
(1) L
(2) L <addr1>
(3) L <addr1> /<n>
(4) L <addr1> <addr2>
```

Comments:

Suppose that you have typed in a segment of instructions or an entire program by using the A command. As we mentioned before in section 5.3.1, the monitor will automatically assemble them into machine codes. At this point, you may wonder if the monitor assembled your program correctly. You are recommended to utilize the L command to have the monitor disassemble all of the converted machine codes into their original instructions.

This command helps you translate as many bytes of machine codes stored in memory as you want into their corresponding original 8088 assembly instructions. For convenience, we provide this command with a default amount of 32 bytes of machine codes to be converted, please refer to syntax 1 that follows.

In all cases, the amount of bytes of machine codes disassembled and displayed on the screen may be slight more than the one requested or the default one. This is because instructions are of variable length. Thus, when this command ends up with the first byte of the last instruction requested, it will disassemble the rest of bytes that instruction owns as well.

Moreover, be sure that the address parameters point to memory locations containing valid 8088 machine codes. If you specify an address that does not contain the first byte of a valid instructin, the display on the screen will be erroneous.

addr#: Each address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

n: Represents the number of bytes of machine codes to be disassembled. It is a hexadecimal number.

Syntax 1

This command will translate into assembly instructions the default 32 bytes of machine codes from the address pointed to by the internal pointer used by the monitor.

Syntax 2

This command will translate into assembly instructions the default 32 bytes of machine codes from the address pointed to by the address parameter <addr1> specified in the command.

Examples:

1. Assume that the following instructions and their corresponding machine codes exist in memory:

```
Instructions:                         Machine Codes:

ØØ8Ø:                                 ØØ8Ø:
ØØØØ   POP   AX                       ØØØØ   58 BØ FF BA
ØØØ1   MOV   AL,ØFF                   ØØØ4   6Ø Ø1 EE BØ
ØØØ3   MOV   DX,Ø16Ø                  ØØØ8   Ø7 Ø8 D8 BA
ØØØ6 · OUT   DX,AL                    ØØØC   8Ø Ø1 EE BA
ØØØ7   MOV   AL,7                     ØØ1Ø   CØ Ø1 EC DØ
ØØØ9   OR    AL,BL                    ØØ14   C8 ØØ ØØ ØØ
ØØØB   MOV   DX,Ø18Ø
ØØØE   OUT   DX,AL
ØØØF   MOV   DX,Ø1CØ
ØØ12   IN    AL,DX
ØØ13   ROR   AL,1
```

Assuming that the internal pointer used by the disassemble command contains ØØ8Ø:ØØØØ, enter the following command: (before entering the command that follows, you are recommended to read section 5.3.14 for the P command to adjust the speed of displaying on the screen.)

```
>L
```

The system will respond with

```
ØØ8Ø:
ØØØØ POP    AX
ØØØ1 MOV    AL,FF
ØØØ3 MOV    DX,Ø16Ø
ØØØ6 OUT    DX,AL
ØØØ7 MOV    AL,Ø7
ØØØ9 OR     AL,BL
ØØØB MOV    DX,Ø18Ø
ØØØE OUT    DX,AL
ØØØF MOV    DX,Ø1CØ
ØØ12 IN     AL,DX
ØØ13 ROR    AL,1
ØØ15 ADD    [BX+SI],
     AL
ØØ17 ADD    [BX+SI],
     AL
ØØ19 ADD    [BX+SI],
     AL
ØØ1B ADD    [BX+SI],
     AL
ØØ1D ADD    [BX+SI],
     AL
ØØ1F ADD    [BX+SI],
     AL
```

↵ The valid machine language codes of this program only takes up 21 bytes, since the L command disassembles 32 bytes of machine codes, the remaining 12 bytes are also disassembled. Because the contents of the unused memory locations in system RAM are set to zeroes during system initialization, and zeroes in consecutive memory locations are disassembled to the instruction ADD [BX+SI],AL, the remaining 12 bytes are all disassembled to this instruction.

2. Taking the assumption of example 1 for instance, enter the following command:

   ```
   >L 0080:0006
   ```

   or

   ```
   >L 80:6
   ```

   The system will respond with

   ```
   0080:
   0006 OUT    DX,AL
   0007 MOV    AL,07
   0009 OR     AL,BL
   000B MOV    DX,0180
   000E OUT    DX,AL
   000F MOV    DX,01C0
   0012 IN     AL,DX
   0013 ROR    AL,1
   0015 ADD    [BX+SI],  <-- Not part of our program
        AL
   0017 ADD    [BX+SI],           "
        AL
   0019 ADD    [BX+SI],           "
        AL
   001B ADD    [BX+SI],           "
        AL
   001D ADD    [BX+SI],           "
        AL
   001F ADD    [BX+SI],           "
        AL
   0021 ADD    [BX+SI],           "
        AL
   0023 ADD    [BX+SI],           "
        AL
   0025 ADD    [BX+SI],           "
        AL
   ```

3. Once again based on the assumption of example 1, enter the following command:

   ```
   >L 80:0 /14
   ```

   The system will respond with

   ```
   0080:
   0000 POP    AX
   0001 MOV    AL,FF
   0003 MOV    DX,0160
   0006 OUT    DX,AL
   0007 MOV    AL,07
   0009 OR     AL,BL
   000B MOV    DX,0180
   000E OUT    DX,AL
   ```

```
    ØØØF MOV     DX,Ø1CØ
    ØØ12 IN      AL,DX
    ØØ13 ROR     AL,1
```

4. Once again based on the assumption of example 1, enter the following command:

   ```
   >L 8Ø:Ø 8Ø:14
   ```

   The system will respond with

   ```
   ØØ8Ø:
   ØØØØ POP     AX
   ØØØ1 MOV     AL,FF
   ØØØ3 MOV     DX,Ø16Ø
   ØØØ6 OUT     DX,AL
   ØØØ7 MOV     AL,Ø7
   ØØØ9 OR      AL,BL
   ØØØB MOV     DX,Ø18Ø
   ØØØE OUT     DX,AL
   ØØØF MOV     DX,Ø1CØ
   ØØ12 IN      AL,DX
   ØØ13 ROR     AL,1
   ```

Part B — Program Execution Commands

5.3.3 Command G — Execute a Program

Name: GO

Purpose:, To execute a program in memory

Syntax:

```
(1) G
(2) G <addr>
```

Comments:

addr: The address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

If no address is specified as in syntax 1 above, the monitor will use the current contents of the CS register as the segment value and the current contentss of the instruction pointer (referred to as IP) to calculate the address at which to begin execution. Execution will continue until a breakpoint is encountered or until the program terminates.

Examples:

1. Suppose that the user code segment register CS contains 0080 and the instruction pointer IP contains 0000, and let us example 1 of the A cmmand to ahead with the following description, The command

```
>G
```

will cause execution to begin from the address 0080:0000 and then the program will print "MICROPROFESSOR I/88".

2. Suppose that the user CS contains 0080. Let us use Example 2 of the A command to go ahead with the following description. The command

```
>G 20
```

will cause execution to begin from the address 0080:0020 and then the program will print "MICROPROFESSOR I/88".

3. Let us use example 3 of the A command to go ahead with the following description. The command

```
>G 90:0
```

or

```
>G CS:0
```

will cause execution to begin from the address 0090:0000 and then print the message as defined in the program.

5.3.4 Command S — Execute a Program step by step

Name: STEP

Purpose: To singel-step a program or execute a specified number of instructions and then stop with a display of register contents on the screen; execution starts from the address pointed to by the code segment CS register and the IP instruction pointer.

Syntax:

```
(1) S
(2) S n
```

Comments:

If no parameter is specified in the S command as listed in syntax 1 above, then the monitor assumes that the program is to be single-stepped.

Note that in the command syntax 2 listed above, the parameter n specifies the number of instructions which will be executed after the S command has been entered.

Before using this command, be sure that the contents of the CS and IP registers point to the address you desire. Please refer to the description of the X command given in section 5.3.7 for instructions on displaying and altering register contents.

After an instruction or the specified number of instructions have been executed, the program will be halted and register contents will be displayed. Because of the screen size, only two registers and the next instruction to be executed will be displayed at a time. At this time, you can type CTRL-A or CTRL-Z to scroll the screen. Typing S will cause program execution to begin from the instruction which is displayed on the last line of the logical screen and the dsiplayed register contents to to updated. Pressing the return key will return you to the monitor prompt ">".

There are 14 registers that can be displayed on the screen: **AX, BX, CX, DX, SI, DI, IP, SP, FL(flag), BP, CS, DS, ES, and SS.**

Every time you enter the S command, the green LED to the upper left of the keyboard will flash once to indicate that the monitor has accepted it and executed an instruction.

Examples:

1. Suppose that CS=0080 and IP=0000. Let us use again the sample program in chapter three to go ahead with the following description. As long as you press the S key with the carriage return key at the keyboard as follows:

   ```
   >S ↵
   ```

   the system will respond with the following messages:

   ```
   AX=0000     BX=0000
   CX=0000     DX=0000
   ```

2. At this point, assume that you press CTRL-Z; then the system will display the contents of the next four registers as follows:

   ```
   SI=0000     DI=0000
   IP=0003     SP=0800
   ```

3. If you press CTRL-Z again, the screen will display the following:

```
    FL=F002      BP=0000
    CS=0080      DS=0080
```

4. Press CRTL-Z again, the screen will display the following:

```
    CS=0080      DS=0080
    ES=0080      SS=0080
```

5. Press CTRL-Z again, the screen will display the following:

```
    ES=0080      SS=0080
   0003 MOV       CX,0064   (Instruction to be executed next)
```

6. Now, if you press CTRL-A 3 times and type S again, the screen will show:

```
   SI=0000      DI=0000
   IP=0006      SP=0800
```

Part C — Debugging

5.3.5 Command B — Set/Show Breakpoints

Name: **BREAKPOINT**

Purpose: To set up to three breakpoints or display their current settings. When a program is on execution and runs into a breakpoint address, the program execution will be halted.

Syntax:

```
(1) B
(2) B <n>
(3) B <n> <addr>
```

Comments:

This command allows you to set up to three addresses in your program as breakpoints. It can also display the breakpoints you have set currently in a program. Breakpoints are a very useful debugging tool that allows you to stop your program at specific locations to allow you to examine registers and memory contents.

n: The breakpoint number, ranging from 1 to 3.

addr: An address at which the breakpoint is to be set. Please note that breakpoints cannot be set at addresses in ROM memory. By the way, the address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

Examples:

1. >B 1 9Ø:9

   This command sets breakpoint one at the address ØØ9Ø:ØØØ9.

2. Assuming that the CS register contains ØØ8Ø, entering the following command:

   >B 2 1B

   will set breakpoint two at the address ØØ8Ø:ØØ1B.

3. >B

   This command will display all of the breakpoints you have set currently in the system, such as the followings:

```
Ø1 ØØ9Ø:ØØØ9
Ø2 ØØ8Ø:ØØ1B
```

   Whenever you forget what breakpoints are in the system, use this command to remind you.

Assume that there is a breakpoint set at ØØ8Ø:ØØØ4. If you started execution from location ØØ8Ø:ØØØØ (and the program didn't first encounter some kind of branch instruction) you will see the following message on the screen.

```
Break at ØØ8Ø:ØØØ4
>█
```

At this point, execution is halted and the monitor is at the prompt level waiting for you to type a command. You could type "G" to resume execution (remember that CS and IP will be pointing to the next instruction that was to be executed when the breakpoint was encountered), "S" to do step execution for debugging, or "X" or "M" to examine or change register or memory contents.

5.3.6 Command C — Cancel Breakpoints

Name: CANCEL

Purpose: To cancel one or all of the breakpoints set previously.

Syntax:

```
(1) C
(2) C <n>
```

Comments:

The command can be used to cancel one or all of the breakpoints you have previously set in the system. If you are finished with debugging and want your program to run without interruption, you can use this method.

n: It is a number ranging from 1 to 3.

Syntax 1

This command will cancel all the breakpoints set currently in a program.

Syntax 2

This command will cancel a breakpoint specified by <n> following the C command.

Examples:

1. To cancel all the breakpoints set currently, enter:

   ```
   >C
   ```

   The system will cancel the breakpoints and return you to the monitor prompt ">".

2. To cancel breakpoint three set currently in the system, enter:

   ```
   >C 3
   ```

   The system will cancel the specified breakpoint and return you to the monitor prompt ">".

5.3.7 Command X — Examine/Alter the Contents of Registers

Name: REGISTER

Purpose: To display or change the contents of any of the registers.

Syntax:

```
(1) X
(2) X <registername>
```

Comments:

Syntax 1

This version of the X command allows you to display the contents of all the registers. After you enter the X command, the contents of four registers will be displayed. By pressing CTRL-Z, you can see the next four. Pressing CTRL-A will bring back the previous four. By pressing the return key, you can go back to the monitor prompt.

All together there are 14 registers that can be displayed: AX, BX, CX, DX, SI, DI, IP, SP, FL (flag), BP, CS, DS, ES, and SS.

Syntax 2

This version of the X command displays the value of a single register and gives you the option of replacing its contents with a value you are to type in. See example 2 below for details.

The valid <registernames> are:

AX	BX	CX	DX
SI	DI	IP	SP
FL	BP	CS	DS
ES	SS		

You can use this command to change the contents of any register among the above 14 ones.

Examples:

1. Show the contents of registers by entering the X command with the carriage return key, such as:

```
>X
```

The system might respond with:

```
AX=0000      BX=0001
CX=0000      DX=FFFF
```

Note:

Depending on the current state of the system. You will probably see different numbers.

At this point, you can either press CTRL-Z and CTRL-A to display other registers or else press the carriage return key to leave the X command and return to the monitor.

2. Assume that the CS register currently contains 0080 and that you want to replace it with 0090. Enter the following command:

```
>X CS
```

The system will respond by displaying the register name and its current contents.

```
CS=0080 █
```

The cursor (█) is staying on the same line. You can type in a number, in this case 0090 or 90, and press the return key to change the contents of the CS register. Or, if you decide not to change the value, you can press the return key without typing anything and the CS register will be left unchanged.

5.3.8 Command M — Examine/Change the Contents of memory

Name: **MEMORY**

Purpose: To display or change the contents of a memory location or a range of memory locations.

Syntax:

```
(1) M
(2) M <addr1>
(3) M <addr1> <addr2>
(4) M <addr1> <addr2> /<data1>/
(5) M <addr1> /<data1> [data2] ..../
(6) M <addr1> <addr2> /<data1> [data2] ..../
```

Comments:

The functions of this command can be divided into three categories:

1. display (examine) or change the contents of a memory location.

2. display (examine) or change the contents of

a range of memory locations.

3. fill the memory locations with values.

addr#: Each address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

data#: Represents any alphanumeric character or hexadecimal number. If you use character data, they should be prefixed with an apostrophe ('), such as 'ABCD. However, if you use hexadecimal data, the apostrophe is not needed.

Examples:

1. Assuming that the current internal pointer contains ØØ8Ø:ØØØØ, enter the following command:

```
>M
```

The system will automatically respond with the contents of eight consecutive memory locations starting from the address ØØ8Ø:ØØØØ for you to inspect as shown below: (Assume those memory locations contain zeros)

```
ØØ8Ø:
ØØØØ ØØ ØØ ØØ ØØ
ØØØ4 ØØ ØØ ØØ ØØ
>
```

At this point, the contents of the internal pointer is ØØ8Ø:ØØØ8. If you enter the same command as above, the screen will show:

```
ØØ8Ø:
ØØØ8 ØØ ØØ ØØ ØØ
ØØØC ØØ ØØ ØØ ØØ
```

2. Assume that the current segment address of the internal pointer is ØØ8Ø, and you intend to alter the contents of memory location ØØ8Ø:ØØØA into 2B, enter: (Refer to syntax 2)

```
>M A
```

The system will respond with

```
ØØ8Ø:
ØØØA ØØ. █
```

and then keeps waiting for you to enter a value, in this case 2B. Pressing the return key following the value you desired will return you to the monitor. However, if you press the space bar instead of the return key, then the system will display the contents of the next memory location as the form shown above and waits for your decision.

3. Assuming that you want to take a look at a certain portion of memory locations, enter: (Refer to the syntax 3)

   ```
   >M ØØ8Ø:ØØØØ ØØ8Ø:ØØØB
   ```

   or

   ```
   >M 8Ø:Ø 8Ø:B
   ```

   or if the contents of the internal pointer segment address equal ØØ8Ø

   ```
   >M Ø B
   ```

   The system will display the contents of 12 consecutive memory locations as follows: (Assume those memory locations contain zeros)

   ```
   ØØ8Ø:
   ØØØØ ØØ ØØ ØØ ØØ
   ØØØ4 ØØ ØØ ØØ ØØ
   ØØØ8 ØØ ØØ ØØ ØØ
   >
   ```

4. Assuming that you want to fill a portion of memory locations with a particular value, enter: (suppose the contents of the internal pointer segment address equal ØØ8Ø)

   ```
   >M Ø F /61/
   ```

   or

   ```
   >M Ø F /'a'/
   ```

   The system will store the hexadecimal value 61 in the memory locations from ØØ8Ø:ØØØØ to ØØ8Ø:ØØØF

5. Assuming that you want to store one or more bytes of data into a range of memory locations staring from the address specified in the command, enter as follows:

   ```
   >M 9Ø:Ø /33 44/
   ```

   This command will have the following effect (but will not cause the result to be displayed on the screen):

   ```
   ØØ9Ø:
   ØØØØ 33 44 ØØ ØØ
   ØØØ4 ØØ ØØ ØØ ØØ
   ```

6. Using the M command, you can also store a particular value or a set of values into a defined range of memory locations. If

there are less values than the specified range of memory locations, the set of values will be used repeatedly until the end of the specified memory locations. Enter as follows:

```
>M 1ØØ:Ø 1ØØ:F /33 44/
```

This command will have the following effect (but will not cause the result to be displayed on the screen):

```
Ø1ØØ:
ØØØØ 33 44 33 44
ØØØ4 33 44 33 44
ØØØ8 33 44 33 44
ØØØC 33 44 33 44
```

Part D — Examing or Altering Memory of Registers

5.3.9 Command I — Insert Data into Memory Locations

Name: INSERT

Purpose: To insert data into a memory location or a portion of memory locations.

Syntax:

```
(1) I
(2) I /<data1> [data2] ..../
(3) I <addr1>
(4) I <addr1> /<data1> [data2] ..../
(5) I <addr1> <addr2>
(6) I <addr1> <addr2> /<data1> [data2] ..../
```

Comments:

You can use this command to insert data into memory. After you have performed the operation of insertion, the contents of the memory locations following the inserted data will be shifted backwards automatically by the length of data inserted.

addr#: Each address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

data#: represents any alphanumeric character or hexadecimal number. If you use character data, they should be prefixed with an apostrophe ('), such as 'ABCD. However, if you use hexadecimal data, the apostrophe is not needed.

Syntax 1

This command will insert one byte with the value zero into the memory location whose address is pointed to by the internal pointer of the current segment. All the original data following the inserted data will be shifted backwards one byte.

Syntax 2

This command will insert one or more bytes of data into a portion of memory locations starting from the address pointed to by the internal pointer of the current segment. All the original data following the last inserted data will be shifted backwards by the length of data inserted.

Syntax 3

With the exception of specifing an address, this command is identical to the syntax 1 in function. You can use this command to insert one byte of data with the value zero into a memory location specified by the address <addr1> in the command. All the original data following the inserted data will be shifted backwards one byte.

Syntax 4

With the exception of specifing an address, this command is much similar to the syntax 2 in function. You can use this command to insert one or more bytes of data into a portion of memory locations starting with the address specified by <addr1> in the command. All the original data following the last inserted data will be shifted backwards by the length of data inserted.

Syntax 5

Same as syntax 1 in function, this command inserts one byte of data with zero value into memory location specified by the first address <addr1> in the command. The original contents of the memory locations up to and including the second address <addr2> will be shifted backwards one byte.

Syntax 6

Same as the syntax 2 in function, this command allows you to insert one or more bytes of data into a portion of memory locations starting with the address specified by the first address <addr1> in the command. The original contents of the memory locations up to and including the second address <addr2> will be shifted backwards by the length of data inserted.

Examples:

1. Assume that the following data exists:

```
ØØ8Ø:
ØØØØ   41 42 43 44
ØØØ4   ØØ 9Ø 9Ø ØØ
ØØØ8   ØØ ØØ ØB ØØ
```

Assuming that the internal pointer contains 0080:0000, enter the following command:

```
>I
```

The contents of these memory locations will be changed as shown below:

```
0080:
 0000  00 41 42 43
 0004  44 00 90 90
 0008  00 00 00 0B
```

2. Taking the assumption of example 1 for instance, enter the following command:

```
>I /'VW' 58 59
```

The contents of these memory locations will be changed as shown below:

```
0080:
0000  56 57 58 59
0004  41 42 43 44
0008  00 90 90 00
000C  00 00 0B 00
```

3. Assume that the following data exists:

```
0080:
0000  30 31 32 33
0004  41 42 43 44
0008  00 24 00 38
```

Enter the following command:

```
>I 0080:4
```

The contents of these memory locations will be changed as shown below:

```
0080:
0000  30 31 32 33
0004  00 41 42 43
0008  44 00 24 00
```

4. Taking the assumption of example 3 for instance, enter the following command:

```
>I 0080:4 /A3 A4 A5/
```

The contents of these memory locations will be changed as shown below:

```
0080:
0000  30 31 32 33
0004  A3 A4 A5 00
0008  41 42 43 44
000C  00 24 00
```

5. Once again using the the assumption of example 3, enter the following command:

   >I 0080:4 0080:7

   The contents of these memory locations will be changed as shown as below:

```
0080:
0000  30 31 32 33
0004  00 41 42 43
0008  00 24 00 38
```

6. Once again based on the assumption of example 3, enter the following command:

   >I 0080:4 0080:7 /D1 C5 C6

   The contents of memory will be changed as shown below:

```
0080:
0000  30 31 32 33
0004  D1 C5 C6 41
0008  00 24 00 38
```

5.3.10 Command D — Delete Data in Memory

Name: DELETE

Purpose: To delete a byte of data or a segmemt of data in memory

Syntax:

(1) D
(2) D /<n>
(3) D <addr1>
(4) D <addr1> /<n>
(5) D <addr1> <addr2>
(6) D <addr1> <addr2> /<n>

Comments:

You can use this command to delete a byte of data or a block of data in memory. After you have performed the operation of deletion, the contents of the memory locations following the deleted data will be shifted forwards automatically by the length of data deleted. However, if you specify a second address in the command, then the data between the end of the range of deletion and the second address will be moved forwards to fill the space. If only one address is specified, all the data from the end of the range of deletion through the end of the current segment will be moved forwards.

addr#: Each address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

n: Represents the number of data bytes you are going to delete. It is a hexadecimal number.

Syntax 1

This command will delete one byte of data whose address is pointed to by the internal pointer of the current segment. All the original data following the deleted data will be shifted one byte forwards.

Syntax 2

This command will delete one or more bytes of data specified by <n> in the command and the the starting address of the data bytes to be deleted is pointed to by the internal pointer of the current segment. After execution of the action of deletion, all the original data following the last deleted data will be shifted forwards by the lenght of data deleted.

Systax 3

With the exception of specifing an address in the command, this command is much similar to the syntax 1 in function. You can use this command to delete one byte of data whose address is specified by <addr1> in the command. All the original data following the deleted data will be shifted forwards one byte.

Syntax 4

With the exception of specifing an address in the command, this command is much similar to the syntax 2 in function. You can use this command to delete one or more bytes of data. The <n> in the command specifies the number of bytes to be deleted and the starting address of the data to be deleted is specified by <addr1> in the command. All the original data following the last deleted data will be shifted forwards by the lenght of data deleted.

Syntax 5

Same as the syntax 1 in function, this command deletes one byte of data whose address is specified by the first address <addr1> in the command. The contents of the memory locations up to and including the second address <addr2> will be shifted forwards one byte.

Syntax 6

Same as the syntax 2 in function, this command allows you to delete one or more bytes of data. The <n> in the command specifies the number of bytes to be deleted and the starting address of data to be deleted is pointed to by the first address <addr1> in the command. The contents of the memory locations up to and including the second address <addr2> will be shifted forwards the length of data deleted.

Examples:

1. Assume that the following data exists:

```
ØØ8Ø:
ØØØØ   41 CD Ø9 EB
ØØØ4   FC B8 ØØ ØØ
ØØØ8   B9 64 ØØ Ø1
ØØØC   ØØ ØØ ØØ ØØ
             :
             :
Ø7FC   ØØ ØA ØB ØC
```

Assuming that the internal pointer of the current segment contains ØØ8Ø:ØØØØ, enter the following command:

```
>D
```

The contents of these memory locations will be changed as shown below:

```
0080:
0000  CD 09 EB FC
0004  B8 00 00 B9
0008  64 00 01 00
000C  00 00 00 00
            :
            :
07FC  0A 0B 0C 0C
```

2. Taking the assumption of example 1 for instance, enter:

   ```
   >D /2
   ```

   The contents of these memory locations will be changed as shown below:

   ```
   0080:
   0000  09 EB FC B8
   0004  00 00 B9 64
   0008  00 01 00 00
   000C  00 00 00 00
               :
               :
   07FC  0B 0C 0B 0C
   ```

3. Taking the assumption of example 1 for instance, enter:

   ```
   >D 80:8
   ```

   The contents of these memory locations will be changed as shown below:

   ```
   0080:
   0000  41 CD 09 EB
   0004  FC B8 00 00
   0008  64 00 01 00
   000C  00 00 00 00
               :
               :
   07FC  0A 0B 0C 0C
   ```

4. Taking the assumption of example 1 for instance, enter:

   ```
   >D 80:8 /4
   ```

The contents of these memory locations will be changed as shown below:

```
0080:
0000   41 CD 09 EB
0004   FC B8 00 00
0008   00 00 00 00
000C   00 00 00 00
             :
             :
07F8   00 0A 0B 0C
07FC   00 0A 0B 0C
```

5. Taking the assumption of example 1 for instance, enter:

```
>D 80:0 80:4
```

The contents of those memory locations will be changed as shown below:

```
0080:
0000   CD 09 EB FC
0004   FC B8 00 00
0008   B9 64 00 01
000C   00 00 00 00
             :
             :
07FC   00 0A 0B 0C
```

6. Taking the assumption of example 1 for instance, enter:

```
>D 80:0 80:5 /3
```

The contents of these memory locations will be changed as shown below:

```
0080:
0000   EB FC B8 EB
0004   FC B8 00 00
0008   B9 64 00 01
000C   00 00 00 00
             :
             :
07FC   00 0A 0B 0C
```

5.3.11 Command F — Find A String of Characters in Memory

Name: **FIND**

Purpose: To search for a specified value or set of values in memory.

Syntax:

```
(1) F /<datastring>
(2) F <addr1> /<datastring>
(3) F <addr1> <addr2> /<datastring>
```

Comments:

This command searches for a specified value or set of values in memory and reports the address where they are found. If the value or values being searched for occur many times in memory, the system will display the address of each occurrence on the screen. You can search for the string you want from the beginning of memory through the end of memory, or just within a specified range of memory, depending on how you enter the command.

addr#: Each address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

datastring:

It represents any alphanumeric character or hexadecimal number. If you use character data, they should be prefixed with an apostrophe ('), such as 'ABCD. However, if you use hexadecimal data, the apostrophe is not needed.

Syntax 1

This command searchs for the value or values specified by <datastring> in the command from the memory location whose address is pointed to by the current CS plus the internal pointer all the way through the end of the current segment.

Syntax 2

This command searchs for a string specified by <datastring> in the command starting from the memory location whose address is specified by the address parameter <addr1> in the command all the way through the end of the current segment.

This command searchs for the value or values specified by <datastring> in the command starting from the memory location whose address is specified by <addr1> in the command through the memory location whose address is specified by <addr2> in the command.

Examples:

1. Assume that the following data exists in memory:

```
ØØ8Ø:
ØØØØ  41 42 43 44
ØØØ4  45 41 42 43
ØØØ8  44 45 41 42
ØØØC  43 44 45 ØØ
ØØ1Ø  ØØ ØØ ØØ ØØ
             :
             :
Ø7F4  ØØ ØØ ØØ ØØ
Ø7F8  41 42 43 44
Ø7FC  45 ØØ ØØ ØØ
```

Enter the following with assuming that the internal pointer of the current segment contains ØØ8Ø:ØØØØ:

```
>F /'AB
```

or

```
>F /41 42
```

The system will respond with:

```
ØØ8Ø:
ØØØØ ØØØ5 ØØØA Ø7F8
```

This means that the hexadecimal values 41 and 42 (representing the ASCII characters 'AB') have been found together at the above four locations. Note that the address given above is the starting address of each occurrence of the string of values.

2. Taking the assumption of example 1 for instance, enter:

```
>F ØØ8Ø:8 /'CD
```

The system will respond with:

```
ØØ8Ø:
ØØØC Ø7FA
```

3. Again taking the assumption of example 1 for instance, enter:

```
>F 0080:8 0080:1A /'E
```

The system will respond with:

```
0080:
0009 000E
```

5.3.12 Command J — Move New Values into the CS and IP registers

Name: **JUMP**

Purpose: To directly jump to the particular address from which you want to start program execution next time.

Syntax:

(1) J <addr>

Comments:

If the address is specified as only an offset, the offset value will replace the contents of the IP register. If both the segment value and the offset value are specified, then they will replace the contents of the CS register and the IP register respectively.

addr: The address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

Examples:

1. Assume that the current CS register contains 0080 and IP contains 000A; and you intend to change the contents of the IP register into 005C. Just enter:

```
>J 005C
```

After execution of this command, the system will retain the original contents of the CS register and alter the contents of the IP register into 005C.

2. Assume that the current CS register contains 0080 and IP contains 0023; and you intend to change the contents of the CS register and the IP register into 0090 and 0000 respectively, Just enter:

```
>J 90:0
```

Note:

You can achieve the same purpose of changing the contents of the CS and IP registers by using the X command, such as the XCS and XIP commands. For detailed descriptions about that both of commands, refer to the X command.

5.3.13 Command T — Copy the Contents of a Memory Range to a Specified Area in Memory

Name: **TRANSFER**

Purpose: To copy a range of memory contents to another area in memory.

Syntax:

(1) T **<addr1> <addr2> <addr3>**
(2) T **<addr4> <addr5>** /**<n>**

Comments:

Syntax 1

You can use this command to copy a range of memory contents starting with <addr1> and ending at <addr2> in the command to a range of memory locations starting at <addr3> in the command.

addr1: the starting address of the range of memory to be copied from.

addr2: the ending address of the range of memory to be copied from.

addr3: the destination address to which the range of memory to be copied.

Syntax 2

This command has the exact same effect as syntax 1. The difference between them is the way in that the ranges of memory affected are defined. In this command, the starting address <addr4> of the range to be copied from, the starting address <addr5> of the area to be copied to, and the number of bytes to be copied are specified.

addr4: the starting address of a number of data bytes to be copied from.

addr5: the destination address to which a number of data bytes to be copied.

n: the number of data bytes to be copied, which is a hexadecimal value ranging from 1 to FF.

Note that "addr#" can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

Examples:

1. Assume that the following data exists in memory:

```
   ØØ8Ø:
   ØØØØ   11 22 33 44
   ØØØ4   55 66 77 88
   ØØØ8   99 AA CC DD
```

   You can copy this entire block of data to memory location ØØ8Ø:ØØØC by issuing the following command:

```
   >T 8Ø:Ø 8Ø:8 8Ø:C
```

   The result is

```
   ØØ8Ø:
   ØØØØ   11 22 33 44
   ØØØ4   55 66 77 88
   ØØØ8   99 AA CC DD
   ØØØC   11 22 33 44
   ØØ1Ø   55 66 77 88
   ØØ14   99 AA CC DD
```

2. Taking the assumption of example 1 for instance, enter:

```
   >T 8Ø:Ø 8Ø:8 /8
```

   The result will be:

```
   ØØ8Ø:
   ØØØØ   11 22 33 44
   ØØØ4   55 66 77 88
   ØØØ8   11 22 33 44
   ØØØC   55 66 77 88
   ØØ1Ø   ØØ ØØ ØØ ØØ
```

   Note that the original contents of the memory locations from ØØ8Ø:ØØØ8 to ØØ8Ø:ØØØB are lost after execution of this command.

Part E — Miscellaneous I/O Commands

5.3.14 Command P — Adjust the Speed of Displaying on the Screen

Name: PAUSE

Purpose: To adjust the speed of displaying on the screen.

Syntax: P <n>

Comments:

You can use this command to adjust the speed of displaying on the screen. The monitor will pause for a specified period of time between displaying each character. You may find this command useful when using the M command or other commands that function to display more than one screenful of data.

n: represents a haxadecimal number ranging from Ø to FF. The larger the number is , the slower the speed of displaying will be. If you specify the number FF, the system will pause about 5 seconds between displaying each character.

Example:

1. If you want the system to display the subsequent characters from this point you are operating at lower speed than the normal one, enter the following command:

```
>P 25
```

The number 25 is considered a reasonable speed number for you to see the subsequent displaying characters. Based on this speed, it will take the system about 5 seconds to respond with a line of messages.

5.3.15 Command N — Input and Display on the Screen One Byte of Data from a Port

Name: INPUT

Purpose: To input and display in hexadecimal one byte of data from the specified port.

Syntax: **N <port_address>**

Comments:

Before using this command, you should make sure that the port address you are going to enter is one which is actually assigned to a device. Refer to the following chapter for a description of the I/O ports and addresses of each deivice.

port_address: represents a hexadecimal number ranging from Ø to FFFF corresponding to a defined port.

Example:

1. Assume that we are going to inspect what the data is at the place that the cursor is occuping right now. Enter the following command:

```
>N 1A3
```

   Assuming that the place that the current cursor position contains a blank, the system will respond with:

```
2Ø
>
```

5.3.16 Command O — Send One or More Bytes of Data to an Output Port

Name: OUTPUT

Purpose: To send one or more bytes of data to a specified output port.

Syntax: O <port_address> /<data>

Comments:

This command can send one or more bytes of data to the specified output port in the command at a time. Refer to the following chapter for a description of the I/O ports and their corresponding port addresses.

port_address: represents a hexadecimal number ranging from Ø to FFFF corresponding to a defined port.

data: represents any alphanumeric characters or hexadecimal numbers. If you use character data, they should be prefixed with an apostrophe ('), such as 'ABCD. However, if you use hexadecimal data, the apostrophe is not needed.

Example:

1. Assume that you want to output the character "A" to the screen, Enter the following command:

```
>O 1A1 /'A'/
```

or

```
>O 1A1 /41/
```

The system will display the character "A" on the screen.

5.3.17 Command E — Assign an External Terminal as the System Console

Name: EXTEND

Purpose: To allow an external terminal to be used as the console for the MPF-I/88

Syntax: E<n>

Comments:

This command allows an external terminal to be used as the console (keyboard and display screen) of the MPF-I/88. There is also an option to return control back to the MPF-I/88's own built-in keyboard and screen. because the way in which the MPF-I/88 communicates with the outside terminal is that an Asynchronous Communications Adapter (ACA) is used as an interface between them, never forget to insert the ACA card into the slot on the left-upper corner of the system board of the MPF-I/88 when using this command. Please refer to the guide, ACA-PC, to install this Asynchronous Communications Adapter.

Before using this command, you have to make sure that the terminal you are going to interface with the machine MPF-I/88 is set to the following communication parameters.

Baud Rate: 9600
Parity: even
Data Bits: 7
Stop bits: 2

n: The value specified for <n> tells what device is to become the system control.

Ø: represents the MPF-I/88 itself.
1: represents a CRT terminal connected to serial port 1.
2: represents a CRT terminal connected to serial port 2.

Example:

1. Assume that you want to interface the system with a CRT terminal, Enter the following command:

   >E1

   Now, you can communicate with the system using your terminal.

5.3.18 Command W — Write Data to Tape

Name: **WRITE**

Purpose: To record the contents of a range of memory on tape.

Syntax: **W <addr1> <addr2> /<file_name>**

Comments:

addr1:	the starting address of the range of data in memory to be recorded on tape. The address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.
addr2:	the ending address of the range of data in memory to be recorded on tape. The address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.
file_name:	The filename can be from 1 to 8 characters long. you can type any filename in small or capital letters. The filename must be prefixed with the apostrophe (').

Note:

Both of the <addr1> and <addr2> must fall within the same segment (address) range; i.e., different segment addresses on the W command line are invalid. For example, typing the "W ØØ8Ø:Ø ØØ9Ø:AØ/'TEST' command into the system will result in an error.

Example:

1. **>W 8Ø:Ø 8Ø:BF /'TEST**

   The contents of these memory locations starting with ØØ8Ø:ØØØØ and ending at ØØ8Ø:ØØBF will be recorded on tape with the filename called "TEST".

5.3.19 Command R — Read Data from Tape

Name: READ

Purpose: to read data from tape.

Syntax:

```
(1) R /<file_name>
(2) R <addr>/<file_name>
(3) R
(4) R <addr>
```

Comments:

file_name: The filename can be from 1 to 8 characters long. you can type any filename in small or capital letters. The filename must be prefixed with the apostrophe (').

addr: represents the staring address of an area in memory from which the file you specify or the system first encounters on tape will be loaded into memory. The address can be specified as either an offset address or a segment address plus an offset. Please refer to section 5.2.1 for a complete discussion.

If the filename is found out by the system during the period of reading, a message will be shown as follows: (assume that the filename we want is called "TEST".)

```
READ FILE:TEST
```

It could be that there are several files on your tape, and the first file might not be the file you want. If the system come across files other than the one you specify, it will ignore them and send out messages as follows:

```
SKIP FILE:xxxxx
```

NOTE:
"xxxxx" represents whatever the filename is of the other file it has encountered.

The system will then go on searching until it finds the filename specified in the command.

Syntax 1

If the system encounters the file you specify, it will load it into memory locations whose starting address is identified by the one you used when recording that file.
Syntax 2

This version of command is much similar to syntax 1 in function, the only difference is that the starting addree of the range of memory to be loaded is specified in the command.

Syntax 3

Based on this command, the system ignores any filename. whatever the file it first encounters on tape will be loaded into memory. The starting address of the range of memory locations to be loaded is specified by the one you used when recording that file.

Syntax 4

Based on this command, what the system performs in function is similar to syntax 3; except that the starting address of the range of memory locations to be loaded is specified by you.

Very Important:

After you have performed any of the above four options, remember to check if the relative address you used in the loaded program,such as the JMPF and CALLF instructions, meets the offset value within the segment address you specify.

5.4 Control Characters

Control characters are entered by pressing a predefined key while holding down the CTRL or ALT key. Each control character performs a specific function.

You have already learned to use some of the control characters such as <ALT-A> and <ALT-Z>. A summary of the control characters and their functions are described below.

* **<CTRL-S>:** Suspends output to the display. Pressing any key resumes output to the display.

* **<CTRL-P>:** This control character is a toggle switch. Entering this control character once turns on the printer, causing the screen output to be sent to the line printer. Entering this control character a second time turns off the printer.

* **<CTRL-X>:** Cancels the current input line to the command buffer.

* **<ALT-Y>:** Returns the system control to the monitor program.

Control Characters for Moving Cursor in the Command Line Buffer:

* **<ALT-E>:** Moves the cursor up one line.

* **<ALT-X>:** Moves the cursor down one line.

* **<ALT-S>:** Moves the cursor to the left one position.

* **<ALT-D>:** Moves the cursor to the right one position.

* **<ALT-F>:** Returns the cursor to where it was located before it was moved.

Control Characters for Scrolling the Screen:

* **<ALT-A>:** Scrolls down the screen.

* **<ALT-Z>:** Scrolls up the screen.

* **<ALT-Q>:** Returns the cursor to where it was located before it was moved.

Chapter 6

Line Assembler

6.1 The Features of the Line Assembler

The advantage of a line assembler is that when you are coding your program in assembly language program, no memory space is required to store your source program; that is, if your source program is large enough, using the line assembler will save lots of memory space for you.

When you are using the line assembler each time you enter an instruction, the line assembler will immediately translate the source code into machine code. The assembly instructions you enter are stored into memory at successive locations, starting with the address specified in the A command. If no address is specified in the A command, the instructions are assembled into memory locations starting at 0080:0000.

The Line Assembler supports standard 8086/8088 assembly language syntax with the following rules:

1. All numeric values entered are assumed to be hexadecimal and can be up to 4 digits long.

2. Prefix mnemonics must be entered in front of the opcode to which they refer. On the MPF-I/88, they should be entered on a separate line, such as:

```
            0080:0005 REP
            0080:0006 MOVSB
```

3. The segment override mnemonics are **CS:**, **DS:**, **ES:**, and **SS:**. They should be entered on the line preceding the instruction they are to modify:

```
            0080:1000 CS:
            0080:1001 MOV AL,[100]
```

   This is equivalent to the **MOV AL,CS:[100]** syntax used in a two-pass assembler.

4. String manipulation mnemonics must explicitly state the string size. For example, **MOVSW** must be used to move word strings and **MOVSB** must be used to move byte strings.

5. The mnemonic for the far return is **RETF**.

6. The line assembler will automatically assemble short, near, or far jumps and calls, depending on the number of displacement byte to the destination address. These can be overridden with the **NEAR** or **FAR** prefix. For example:

```
0080:00A0 JMP A2              ;A 2-BYTE SHORT JUMP
0080:00A2 JMP NEAR A5         ;A 3-BYTE NEAR JUMP
0080:00A5 JMP FAR AA          ;A 5-BYTE FAR JUMP
```

7. When using the line assembler, you cannot use labels to refer to the addresses to which you intend to jump. You must use absolute addresses. For example:

```
0080:0010 ADD AX,CX
0080:0012 LOOP 10
```

8. Sometimes, the line assembler cannot tell whether some operands refer to a word memory location or a byte memory location. This will happen when none of the operands in an instruction are unambiguously either word or byte length values. In this case, the data type must be explicitly stated with the prefix **"WORD PTR"** or **"BYTE PTR"**. The word **"PTR"** may be omitted when using the line assembler. For example:

```
      MOV BYTE PTR [BX+DI],0A9
or    MOV BYTE [BX+DI],0A9

      MOV WORD PTR [BX+DI],0A9
or    MOV WORD [BX+DI],0A9
```

   In each case, the CPU moves the values 0A9H to the address pointed to by the Base and Index registers. The **"BYTE"** or **"WORD"** operator of the line assembler tells the CPU whether the memory location is a byte or word.

9. The line assembler also cannot tell whether an operand refers to a memory location or to an immediate operand. The line assembler uses the common convention that operands enclosed in square brackets refer to memory. For example:

```
      MOV AX,63           ;LOAD AX WITH 63H
      MOV AX,[63]         ;LOAD AX WITH THE CONTENTS
                          ;OF MEMORY LOCATION 63H
```

10. Two common pseudo-codes are also included in the line assembler. The **DB** opcode will assemble byte values directly into memory in successive order. And the **DW** opcode will assemble word values directly into memory in successive order. For example:

```
      DB  10,11,12,13,"TEST REPORT"
      DB  'SELECTED ANSWERS AS FOLLOWS:'
      DB  "THIS IS A QUOTE:'"

      DW  1388,1770,1B58,"BATCH"
```

11. All forms of the register indirect addressings are supported. For example:

```
ADD  BX,34[BP+2][SI-1]
ADD  BX,34[BP+SI+1]
ADD  BX,[34+BP+2+SI-1]
ADD  BX,[BP+SI+35]
POP  [BP+DI]
PUSH [SI]
```

12. All opcode synonyms are supported. For example:

```
LOOPZ  1ØØ
LOOPE  1ØØ

JA     2ØØ
JNBE   2ØØ
```

Chapter 7

Useful Subroutines

7.1 A List of Useful Subroutines

The following is a list of the service routines that you can use to enhance the versatility of your programs or to communicate with external devices. These routines are actually part of the monitor itself, but can also be used to simplify your programs. As you will notice, they are called by using the software interrupt feature of the 8088 microprocessor. You can study the algorithms used for each of these subroutines by looking them up in your Monitor Source Listing Manual. Detailed descriptions of the functions and exact details for calling each service routine are introduced in the following pages.

Interrupt Number	Subroutine Name
INT 7H	END_OF_PGM
INT 8H	CONSOLE_IN
INT 9H	CONSOLE_OUT
INT AH	GET_VALUE
INT BH	OUTLCD
INT CH	BEEP
INT DH	OUTSTRING
INT EH	OUTBYTE
INT FH	ATOB
INT 10H	OUTWORD
INT 11H	SETGETCUR
INT 12H	READ_LINE
INT 13H	RS232DRIVER
INT 14H	PRT_DRIVER
INT 15H	DIAS
INT 16H	CN_IN_STATUS
INT 17H	KEY_QUEUE_FLUSH
INT 18H	SOUND
INT 19H	GET_MEM_SIZE
INT 1AH	TV_DRIVER (Reserved)
INT 1BH	THERMAL_PRT
INT 1CH	ITAPE_READ
INT 1DH	ITAPE_WRITE
INT 1EH	IENABLE_INT
INT 1FH	IDISABLE_INT
INT 20H	Reserved
INT 21H	Reserved
INT 22H	AVAIL_MEM

7.2 Function Description of the Useful Subroutines

In the description below, the field labelled **Input** describes what information the subroutine will expect to find in which registers. When using a subroutine while writing a program, be sure that you load appropriate values into the correct registers before using the INT statement to call the subroutine. The field labelled **Output** tells you in which registers information returned by the subroutine will be placed when it returns control to your main program. You should pay careful attention especially to the information in the field labelled **Registers Affected.** This field tells which registers are altered by the subroutine; if your program has important information in those registers at the point where you call the subroutine, you should be sure to use the PUSH instruction to save the contents of the registers onto the stack before you call the subroutine and the POP instruction to restore the register contents after returning from the subroutine.

7.2.1 INT 7 — END OF PGM

Name: END_OF_PGM   (end of program)

Function Description:

This routine will end the execution of your program and tranfer control of the system back to the monitor. You should call this routine to end each of your programs.

Input: None

Output: None

Registers Affected: None

7.2.2 INT 8 — CONSOLE IN

Name: CONSOLE_IN (input a character from the current console)

Function Description:

The function of this routine is to fetch a character from the keyboard of the console which is currently communicating with the system and then store it into the AL register in the form of ASCII code.

Input: None

Output: AL

Registers Affected: None

7.2.3 INT 9 — CONSOLE OUT

Name: CONSOLE_OUT (output a character to the current console)

Function Description:

The function of this routine is to display in the ASCII form a character stored in the AL register on the console that is currently communicating with the system.

Input: AL - contains the character to be output.

Output: None

Registers Affected: None

7.2.4 INT A — GET VALUE

Name: **GET_VALUE** (get a value from the current console)

Function Description:

The function of this routine is to fetch a value in the form of ASCII code passed from the keyboard of a console that is currently communicating with the system, and then store into the AX register in hexadecimal form. Notice that you have to tell this routine through the AH register what is the maximum number of digits that can be entered on each line of the screen.

Input: AH - holds the value that indicates the maximum number of digits to be entered on each line of the screen before the cursor should be moved to the next line. Remember that the maximum number that the AH can hold is 255.

Output:

(1). AX - holds the converted character you just entered.

(2). CL - holds the actual number of digits you just entered.

(3). ZF=Ø - indicates that the characters you just entered end with the space bar.

(4). ZF=1 - indicates that the characters you just entered end with the return key.

Registers Affected: AX, CX

7.2.5 INT B — OUTLCD

Name: **OUTLCD** (output to LCD)

Function Description:

The function of this routine is to output a character stored in the AL to the LCD screen of the MPF-I/88. This routine can output any character contained in the ASCII set supported by the MPF-I/88.

Input:

(1). AL - holds the character to be outpuւ

(2). CH=Ø - indicates that you expect no cursor will appear on the LCD screen.

CH<>Ø - means the reverse of CH=Ø; i.e., you expect that the cursor will display on the screen.

(3). CL=Ø - indicates that you expect the system not to scroll up the screen.

CL<>Ø - means you expect the system to scroll up the screen.

Output: None

Registers Affected: None

7.2.6 INT C — BEEP

Name: BEEP

Function Description:

The function of this routine is to cause the MPF-I/88 to make a short beep sound from its speaker.

Input: None

Output: None

Registers Affected: None

7.2.7 INT D — OUTSTRING

Name: OUTSTRING (output a string to the current console)

Function Description:

The function of this routine is to output a string of data which is stored in memory in the form of ASCII code to the screen of the console that is currently communicating with the system. This routine doesn't display characters with ASCII codes greater that 8ØH.

Input:

(1). DS - contains the segment address value of the start of the string of data to be output.

(2). SI - contains the offset value of the start of the string of data to be output.

(3). The last byte of the string of data to be output should have the sign bit set to 1, representing the end of the string of data to be output.

Output: None

Registers Affected: None

7.2.8 INT E — OUTBYTE

Name: **OUTBYTE** (output a byte to the current console)

Function Description:

The function of this routine is to display in hexadecimal form the contents of the AL register on the screen of the console which is currently communicating with the system.

Input: AL

Output: None

Registers Affected: None

7.2.9 INT F — ATOB

Name: **ATOB** (convert ASCII numeric string to binary)

Function Description:

The function of this routine is to convert a string of numerical data stored in memory in the form of ASCII code into binary numbers. Note that you have to tell the system through the CX register what is the maximum number of characters you are going to convert.

Input:

(1). CX - holds the maximum number of characters to be converted. Remember that the maximum number that the CX can hold is 65535.

(2). DS - stores the segment address value of the start of the string of characters to be converted.

(3). SI - stores the offset address value of the start of the string of characters to be converted.

Note:

No ending address or any mark to indicate the ending address is required for this routine. This routine will automatically convert the string of data you require until it finds a character that is unconvertible into binary or encounters the length limit in the CX register.

Output: DX

Registers Affected: AX, DX, CX, SI

7.2.10 INT 10 — OUTWORD

Name: **OUTWORD** (output a word to the current console)

Function Description:

The function of this routine is to display in hexadecimal form a character or number stored in the AX register on the screen of the console that is currently communicating with the system.

Input: AX - contains the data to be output.

Output: None

Registers Affected: AX, CX, DX

7.2.11 INT 11 — SETGETCUR

Name: **SETGETCUR** (set or get current cursor position)

Function Description:

The function of this routine is to set or inquire about the current cursor position on the LCD screen of the MPF-I/88.

Input:

(1). AH=Ø - indicates to set the cursor position.

(2). AH=1 - indicates to obtain the current cursor position.

In the case of setting the cursor position:

(3). CH - contains the row number on the LCD screen. The number ranges from Ø to 1.

(4). CL - contains the column number on the LCD screen. The number ranges from Ø to 19.

Output:

In the case of obtaining the current cursor position:

(1). CH - contains the row number on the LCD screen. The number ranges from Ø to 1.

(2). CL - contains the column number on the LCD screen. The number ranges from Ø to 19.

Registers Affected: BX, CX, SI

7.2.12 INT 12 — READ LINE

Name: READ_LINE (read a line from the current console)

Function Description:

The function of this routine is to get a line of data from the keyboard of the console which is currently communicating with the system and also display the line of data on the screen of that console.

Input:

(1). AH - holds the value that indicates the maximum number of characters to be entered on each line of the screen. Remember that the maximum number that the AH register can hold is 255.

(2). DS - holds the segment address of memory in which the data to be entered will be stored.

(3). SI - holds the offset address at which the data to be entered will be stored.

Output: None

Registers Affected: None

7.2.13 INT 13 — RS232 DRIVER

Name: RS232DRIVER (start up the RS232 driver)

Function Description:

The function of this routine is to interface with external devices using an asynchronous (serial) communications interface. There are several different specific functions provided by this routine, such as sending out and fetching data to and from the external devices, examining the status of 825Ø serial controller, or initializing the communication port itself.

Input:

(1). DX - contains the port address of the port RS232 to which the external device is connected. On the MPF-I/88, two port addresses are provided. They are 2F8 and 3F8.

Note that the DX should always be stored with the port address whenever this routine is used no matter which function is to be performed.

A. FUNCTION 1:

Initializes the serial communications port.

(2). AH=Ø - indicates that serial port should be initialized.

AL - Contains the initialization parameters.

Initialization Parameters:

7 6 5	4 3	2	1 Ø
BAUD RATE	PARITY	STOPBIT	WORD LENGTH
ØØØ - 11Ø	1Ø - None	Ø - 1	1Ø - 7 BITS
ØØ1 - 15Ø	Ø1 - ODD	1 - 2	11 - 8 BITS
Ø1Ø - 3ØØ	11 - EVEN		
Ø11 - 6ØØ			
1ØØ - 12ØØ			
1Ø1 - 24ØØ			
11Ø - 48ØØ			
111 - 96ØØ			

B. FUNCTION 2:

Transmits a character which is stored in AL and return status in AH.

AH=1 - indicates that a character should be output through the port.

AL - holds the character to be output.

C. **FUNCTION 3:** To receive a character.

AH=2 - indicates to receive a character from an external device over the communication port.

D. **FUNCTION 4:**

To examine the status of the RS232 port.

AH=3 - indicates to examine the status of an external device over the communication port.

Output:

A. **FUNCTION 1:**

This function returns serial port statuses in AX.

B. **FUNCTION 2:**

If a character is transmitted successfully, this function returns status in AH. Otherwise, bit 7 of AH is set.

C. **FUNCTION 3:**

If a character is received, AL will hold that character. This function returns status in AH.

D. **FUNCTION 4:**

This function returns serial port statuses in AX. AH will contain the line status and AL will contain the modem status.

The meaning of each bit in the AH register holding the status:

```
BIT 7 = Time out
BIT 6 = Trans shift register empty
BIT 5 = Trans holding register empty
BIT 4 = Break detect
BIT 3 = Framing error
BIT 2 = Parity error
BIT 1 = Overrun error
BIT Ø = Data ready
```

The meaning of each bit in the AL register holding the status:

```
BIT 7 = Receive line signal detect
BIT 6 = Ring indicator
BIT 5 = Data set ready
BIT 4 = Clear to send
BIT 3 = Delta receive line signal detect
```

BIT 2 = Trailing edge ring detector
BIT 1 = Delta data set ready
BIT Ø = Delta clear to send

Registers Affected: None except for AX

7.2.14 INT 14 — PRT DRIVER

Name: PRT_DRIVER (printer driver)

Function Description:

The function of this routine is to output data to the Centronics parallel port which is built in the system.

Input: AL - contains the data to be printed.

Output:

AH=Ø - indicates that the data you required was successfully printed out.

AH=1 - indicates that the data you required was not successfully printed out.

Registers Affected: None

7.2.15 INT 15 — DISA

Name: DISA (disassemble)

Function Description:

The function of this routine is to convert a machine language instruction into assembly source mnemonic form.

Input:

(1). ES - contains the segment address of the instruction to be disassembled.

(2). DI - contains the offset address of the instruction to be disassembled.

Output: None

Registers Affected: None

7.2.16 INT 16 — CN IN STATUS

Name: CN_IN_STATUS (check the input status of the current console)

Function Description:

The function of this routine is to check the input status of the console which is currently communicating with the system. Through the result of status sent from the console you can see if the console is busy performing some operation, if any data is ready to be received from the console, and so on.

Input: None

Output:

(1). AL=Ø - indicates that there is no data ready to be received from the console.

(2). AL<>Ø - indicates that there is some data ready to be received from the console.

Note:

If the console currently communicating with the system is an external terminal, then a non-zero status byte in the AL register can be interpreted as follows:

The respective meaning of each bit in the AH register holding the status:

BIT 7 = Time out
BIT 6 = Trans shift register empty
BIT 5 = Trans holding register empty
BIT 4 = Break detect
BIT 3 = Framing error
BIT 2 = Parity error
BIT 1 = Overrun error
BIT Ø = Data ready

The respective meaning of each bit in the AL register holding the status:

BIT 7 = Receive line signal detect
BIT 6 = Ring indicator
BIT 5 = Data set ready
BIT 4 = Clear to send
BIT 3 = Delta receive line signal detect
BIT 2 = Trailing edge ring detector
BIT 1 = Delta data set ready
BIT Ø = Delta clear to send

Registers Affected: None

7.2.17 INT 17 — KEY QUEUE FLUSH

Name: **KEY_QUEUE_FLUSH** (flush the keyboard input queue)

Function Description:

On the MPF-I/88, each time you enter a character at the keyboard, the system stores it in a working area, called the key queue. Whenever you want to clear or ignore the contents of the key queue in the system, just call this routine.

Input: None

Output: None

Registers Affected: None

7.2.18 INT 18 — SOUND

Name: **SOUND**

Function Description:

The function of this routine is to have the MPF-I/88 make a sound from its speaker based on the contents of the registers BX and CX.

Input:

(1). BX - holds the duration of the sound.

(2). CX - holds the frequency of the sound.

Output: None

Registers Affected: None

7.2.19 INT 19 — GET MEM SIZE

Name: **GET_MEM_SIZE** (get memory size)

Function Description:

The function of this routine is to ascertain the size of the RAM.

Input: None

Output:

AX - contains the result of the memory size. The unit of the number is in kilobytes.

Registers Affected: None

7.2.20 INT 1B — THERMAL PRT

Name: **THERMAL_PRT** (thermal printer driver)

Function Description:

The function of this routine is to output data to the thermal printer attached to the system.

Input: AL - contains the data to be printed.

Output: None

Registers Affected: None

7.2.21 INT 1CH —— READ DATA TAPE

```
             Name: ITAPE_READ
         Function: Read data from tape.
            Input: CX  is used to store the number of bytes to be
                   read.
                   The  ES and DI registers are used as  pointers
                   to point to the memory block (buffer area)  to
                   be read.
           Output: If CL is cleared to Ø,  then there is no error
                   during tape read operation.
                   If CL is not cleared to Ø,  then error  occurs
                   during tape read operation.
Register Affected: All
```

7.2.22 INT 1DH —— WRITE DATA TO TAPE

```
             Name: ITAPE_WRITE
         Function: Write data to tape (in IBM PC format).
            Input: CX contains the number of bytes to be written.
                   The  DS and SI registers are used as  pointers
                   to  point to the memory buffer whose  contents
                   are to be written.
           Output: None
Register Affected: All
```

7.2.23 INT 1EH —— IENABLE INT

```
             Name: IENABLE_INT
         Function: Enable keyboard interrupt.
            Input: None
           Output: None
Register Affected: None
```

(Please refer to Chapter Five I/O Device Drivers of MPF-I/88 Reference Manual for more details of the keyboard interrupt.)

7.2.24 INT 1FH —— IDISABLE-INT

```
             Name: IDISABLE_INT
         Function: Disable keyboard interrupt.
            Input: None
           Output: None
Register Affected: None
```

7.2.25 INT 20H and 21H

INT 2ØH and 21H are reserved.

7.2.26 INT 22H —— AVAIL-MEM

Name:	AVAIL_MEM
Function:	Set or display the starting address of free memory space in the user's RAM.

(A) Display (Get) the Starting Address

Input:	AH = Ø
Output:	The starting address is returned in the ES (for segment address) and DI (for offset address) registers.

(B) Set Starting Address

Input:	AH = 1. The starting address is stored in the ES (for segment address) and DI (for offset address) registers.

INT ØFFH

Since some integrated circuits can only generate the interrupt vector "FF", four bytes are reserved for the instruction INT FF as interrupt vector. The default value of the interrupt vector causes INT ØFFH to jump back to command interpreter.

ALT Y

When entering the ALT-Y control character, the monitor program will execute the instruction INT 6. The default vector of INT 6 causes the monitor program to jumpt back to the command interpreter. The INT 6 was intended as a feature which enables the user to transfer the control of a program to a desired location during program execution.

You can change the value of the interrupt vector so that any time ALT Y is entered, program control will be transferred to the desired location.

Appendix A CPU Pin Function Description

The 8Ø88 CPU contains the following pins:

1) The 8-bit system data bus,
2) The 2Ø-pin address bus,
3) Power and ground for interface boards,
4) Clock and timing signals,
5) The control lines for memory or I/O read or write,
6) Maskable and non-maskable interrupt request lines,
7) Status line for Interrupt Enable Flag,
8) Bus cycle status lines.

The signal lines provided by the 8Ø88 CPU are described as follows:

> Note: In the following description of the CPU pin functions, "I" is used to represent that the signal line is an input signal from system bus; "O" represents an output to system bus; while "I/O" indicates a signal which can either be an input or an output line.

$MN/\overline{MX}$ (I)

This pin is used to control whether the 8Ø88 is configured to operate in minimum mode or maximum mode. If the 8Ø88 is configured to operate in maximum mode, it can operate in close interaction with other coprocessors such as the 8Ø87. If this pin is pulled high, then the 8Ø88 operates in minimum mode. If it is strapped low, the CPU operates in maximum mode. Since the MPF-I/88 is configured to operate in minimum mode, this pin is pulled high.

DØ - D7 (I/O)

These lines are connected to the system data bus. These pins are tri-state signal lines. They are used to carry data to and from the CPU.

AØ - A19 (O)

These lines are connected to the system address bus, which can address up to one megabytes of memory. These lines are tri-state signal lines and are generated by the processor.

READY (I)

The READY line is used by slow memory or I/O devices to signal a "not ready" state so that the CPU can insert additional wait states in a bus cycle.

$\overline{\text{TEST}}$ (I)

This pin is sampled when the CPU executes the WAIT instruction. If it is low when tested, the instruction following the WAIT instruction is executed. If it is high, the CPU will pause.

$\overline{\text{RD}}$ (O)

This pin is active low. It is pulsed low to indicate that the CPU is reading data from a memory or I/O device.

$\overline{\text{WR}}$ (O)

This pin is active low. It is pulsed low to indicate that the CPU is writing data to a memory or I/O device.

$\overline{\text{IO}}$/M (O)

This is pulled high for a memory access, and low for an I/O access.

HOLD (I)

This pin is used by an external device, usually a bus master, to request for the service of the system bus. If an external device needs the system bus for data transfer, it will pulse this pin high. Upon detecting a high on this pin, the CPU will complete the current bus cycle and issue a hold acknowledge through the HLDA pin. A HOLD signal from an external device requests the CPU to release the system bus temporarily for servicing the need of the external device.

HLDA (O)

This signal is high active. This signal is raised high when the CPU acknowledges a HOLD request from an external device. Once the HOLD request is acknowledged, the CPU releases bus control to the requesting external device by floating the system bus between itself and the memory and I/O devices in order to let the external device to take over the control of the system bus.

ALE (O)

This pin, Address Latch Enable, is active high, and is generated by the 8Ø88 for latching valid address.

$\overline{\text{DEN}}$ (O)

This signal, data enable, is active low. This signal determines whether the data buffer output is enabled or disabled. Data can be output only when this pin is active.

$\overline{\text{SSØ}}$ (O)

This tri-state output is used with other two signal lines to provide bus cycle status to the peripheral devices. The other two signals which determine bus cycle status are $\overline{\text{IO}}$/M and $\overline{\text{DT}}$/R.

DT/$\overline{\text{R}}$ (O)

In addition to providing bus cycle status, the 3-state output line determines the direction of data transfer between the CPU and data buffer. If this pin is raised high, data is transferred from the CPU to the data buffer. If this pin is strapped low, data is transferred from the data buffer to the CPU.

S3, S4 (O)

These two signal lines are multiplexed with address lines A16 and A17. These two pins determine which segment register provides the segment portion of an actual address. During all T (clock period) states except for the first T state, information on the segment portion of the actual physical address is present on the output of these two pins. In the first T state, these two pins are used as ordinary address lines - A16 and A17. In an I/O bus cycle, these pins are always low in the first clock period.

S5 (O)

This line is multiplexed with the address line A18. During the first clock period, this pin serves as A18. During all other clock periods, this pin is used to reflect the state of the CPU's Interrupt Enable flag. In an I/O bus cycle, this pin is always low in the first clock period.

S6 (O)

This line is multiplexed with the address line A19. During the first clock period, this pin serves as A19. In an I/O bus cycle, this pin is always low in the first clock period. During all other clock periods, this pin is strapped low if the system bus is under the control of the CPU. However, if the system bus is under the control of other bus masters such as a DMA controller, this pin is floated by the CPU to allow the other bus master to gain control of the system bus.

INTR (I)

This line is used by the I/O devices to generate interrupt requests to the CPU. When an interrupt request is generated, this pin is held high until the processor acknowledges.

NMI (I)

This line is used by the I/O devices to generate interrupt requests to the CPU. When an interrupt request is generated, this pin goes from low to high (edge-triggered).

$\overline{\text{IOR}}$ (O)

This control signal, active low, causes the an addressed I/O device to present data onto the data bus. It is driven by the processor.

$\overline{\text{IOW}}$ (O)

This control signal, active low, causes the addressed I/O device to read the data present on the data bus. It is driven by the processor.

$\overline{\text{MEMR}}$ (O)

This control signal, active low, causes the addressed memory device to present data onto the data bus. It is driven by the processor.

$\overline{\text{MEMW}}$ (O)

This control signal, active low, causes the addressed memory device to fetch the data present on the data bus. It is driven by the processor.

CLK (O)

System clock. It is derived by dividing the oscillator by three. It has a clock period of 21Ø ns and 33% duty cycle.

Appendicx B I/O Expansion Bus Pin Function Description

The I/O expansion bus extends the functions of the 8088 system bus with the addition of signal pins for handling interrupt processing.

The I/O expansion bus contains the followings:

1) The 8-bit system data bus,
2) The 20-pin address bus,
3) Power and ground for interface boards,
4) Clock and timing signals,
5) The control lines for memory or I/O read or write,
6) Maskable and non-maskable interrupt request lines,
7) Status line for Interrupt Enable Flag,
8) Bus cycle status lines,

These signal lines are provided in a 62-pin connector.

An I/O ready line -- $\overline{\text{I/O CH RDY}}$ -- is provided on the I/O channel, enabling the processor to perform data transfer with slow memory or I/O devices. If the I/O ready line is not activated by an selected I/O device, all processor read or write cycle takes four clock periods with each clock period being 210 ns. All processor I/O read or write cycle takes five clock periods. Each DMA data transfer takes five clock periods per byte.

If you want to attach external devices to the MPF-I/88, you have to ensure that the system board has sufficient power to drive the external card connected to the system board. The power adapter which is shipped to you together with the MPF-I/88 supplies 1A. A switching power supply that supplies 3A can be used if you intend to insert more than one external devices (peripheral cards) to the system. In case you want to install more than one peripheral cards to the system, you have to calculate the total amount of electric current that the peripheral cards may consume. If the total amount of electric current that may be consumed by the peripheral cards is larger than that is supplied by the power supply, the system will not be able to drive the peripheral cards, and will stop after power-on.

The I/O expansion bus signal lines are listed as follows:

B1	GND	A1	*(-HOLD)
B2	+RESET DRV	A2	+D7
B3	+5V	A3	+D6
B4	+IRQ2	A4	+D5
B5	*(-INTA)	A5	+D4
B6	+DRQ2	A6	+D3
B7	-5V	A7	+D2
B8	*(+INTR)	A8	+D1
B9	12V	A9	+DØ
B1Ø	GND	A1Ø	+I/O CH RDY
B11	-MEMW	A11	+AEN
B12	-MEMR	A12	+A19
B13	-IOW	A13	+A18
B14	-IOR	A14	+A17
B15	-DACK3	A15	+A16
B16	+DRQ3	A16	+A15
B17	-DACK1	A17	+A14
B18	+DRQ1	A18	+A13
B19	-DACKØ	A19	+A12
B2Ø	CLOCK	A2Ø	+A11
B21	+IRQ7	A21	+A1Ø
B22	+IRQ6	A22	+A9
B23	+IRQ5	A23	+A8
B24	+IRQ4	A24	+A7
B25	+IRQ3	A25	+A6
B26	-DACK2	A26	+A5
B27	+T/C	A27	+A4
B28	+ALE	A28	+A3
B29	+5V	A29	+A2
B3Ø	+OSC	A3Ø	+A1
B31	+GND	A31	+AØ
B32	**(+GND)	A32	**(+5V)

NOTE: The asterisk (*) is marked for the signal lines which are assigned different functions as compared with the pin assignment for IBM PC I/O channel. For example, pin B5 is assigned for -INTA for the MPF-I/88 expansion bus rather than -5VDC as defined in the IBM PC expansion slot:

PIN	MPF-I/88	IBM PC
B5	-INTA	-5VDC
B3	+INTR	Reserved
A1	-HOLD	-I/O CH CK

The signal lines provided by the I/O channel are described as follows:

> Note: In the following description of the CPU pin functions, "I" is used to represent that the signal line is an input signal from system bus; "O", an output to system bus; while "I/O", a signal which can either be an input or an output line.

DØ - D7 (I/O)

These lines are connected to the system data bus. These pins are tri-state.

AØ - A19 (O)

These lines are connected to the system address bus, which can address up to one megabytes of memory. These lines are tri-state signal lines and are generated by the processor.

ALE (O)

This pin, Address Latch Enable, is generated by the processor for latching valid address. This line is active high. Note that this pin is an input on the I/O channel. It is, however, an output pin on the CPU.

HOLD (I)

This pin is used by the external devices, usually bus masters, to request control of the system bus. If an external device needs the system bus for data transfer, it will pulse this pin high. Upon detecting a high on this pin, the CPU will complete the current bus cycle and issue a hold acknowledge through the HLDA pin. A HOLD signal from an external device requests the CPU to release the system bus temporarily for servicing the need of the external device. Since HOLD request is sent from an external device through the I/O channel to the CPU, therefore, this pin is an output from the I/O channel, and an input pin on the CPU.

INTR (I)

This line is used by the I/O devices to generate interrupt requests to the CPU. When an interrupt request is generated, this pin is held high until the processor acknowledges. Since interrupt request is sent from external devices via the I/O channel to the CPU, hence this pin is an output from the I/O channel and an input on the CPU.

$\overline{\text{IOR}}$ (O)

This control signal, active low, causes the addressed I/O device to present data onto the data bus. It is driven by the processor. This state of this input is determined by the states of the $\overline{\text{IO}}$/M and the $\overline{\text{RD}}$ pins of the CPU.

$\overline{\text{IOW}}$ (O)

This control signal, active low, causes the addressed I/O device to read the data present on the data bus. It is driven by the processor. The state of this input is determined by the states of the $\overline{\text{IO}}$/M and the $\overline{\text{WR}}$ pins of the CPU.

$\overline{\text{MEMR}}$ (O)

This control signal, active low, causes the addressed memory device to present data onto the data bus. It is driven by the processor. This pin becomes active when a low is present on both the $\overline{\text{IO}}$/M and the $\overline{\text{RD}}$ pins of the CPU.

$\overline{\text{MEMW}}$ (O)

This control signal, active low, causes the addressed memory device to fetch the data present on the data bus. It is driven by the processor. This pin becomes active when a low is present on both the $\overline{\text{IO}}$/M and the $\overline{\text{WR}}$ pins of the CPU.

CLK (O)

System clock. It is derived by dividing the oscillator output by three. It has a clock period of 210 ns and 33% duty cycle. This signal is sent by the CPU to the I/O channel.

$\overline{\text{I/O CH RDY}}$ (I)

The signal (usually high or "ready") is pulled low by a slow memory or I/O device to extend memory or I/O bus cycles. It enables a user to attach slower memory or I/O devices to the I/O channel. This line is driven low by a slow device immediately after it detects a valid address and a read or write command. This line should never be driven low for more than 10 clock periods. This signal is sent from an external device via the I/O channel to the CPU. Therefore, this pin is an output pin from the I/O channel and an input on the CPU.

AEN (O)

This line, active high, enables an external bus master such as a DMA Controller to gain control of the system address bus, data bus, read command lines for a memory or I/O device, or write command lines for a memory or I/O device. Output from the HLDA pin from the CPU is sent to the external bus master via the I/O channel. Therefore, this pin is an input.

OSC (O)

This signal is provided by a oscillator operating at 14.31818 MHz. It has a 5Ø% duty cycle.

RESET DRV (O)

This signal, Reset Driver, is active high. It is used to initialize or reset system logic upon power-up. It is synchronized to the falling edge of clock.

-INTA

This line is used by the CPU to generate an interrupt acknowledge signal to the interrupting device.

> The following signal lines are valid only when external expansion cards built with these signal lines are plugged to the main system board.

IRQ2 - IRQ7 (I)

These lines are used by the I/O devices to generate interrupt requests 2 to 7 to the CPU. They are prioritized with IRQ2 having the highest priority and IRQ7 the lowest priority. When an interrupt request is generated, this pin is held high until the processor acknowledges.

DRQ1 - DRQ3 (I)

This lines, active high, are used by peripheral devices to gain DMA service. These signals are prioritized with DRQ1 having the highest priority and DRQ3 the lowest priority. A DMA request is issued by raising the DRQ line high until the corresponding DACK line becomes active.

$\overline{\text{DACKØ}}$ - $\overline{\text{DACK3}}$ (O)

These lines, active low, are used to acknowledge DMA requests and to refresh system dynamic RAM.

T/C (O)

Terminal Count. This line, active high, sends a pulse when the terminal count for a DMA transfer is reached.

The rest of the I/O channel connector pins are used to provide power and ground to the channel.

Appendix C AUTO ROM and EXTERNAL COMMANDS

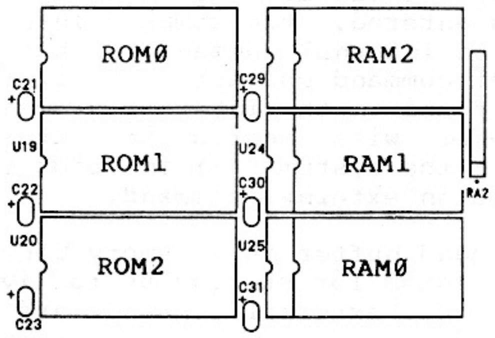
AUTO ROM

The monitor program (version 1.1) of MPF-I/88 allows you design your own auto ROMs for the applications you desire. Program stored in the auto ROM gets executed as soon as you power up the machine. The auto ROM can be inserted in any of the ROM sockets (or the ROM socket on a system expansion card).

Under monitor program version 1.1, an auto ROM can be installed on any of the two built-in ROM sockets on the MPF-I/88 mother board. Board locations U19 and U2Ø are reserved for system ROM expansion.

Take note that you can not install the auto ROM this way under monitor program 1.Ø.

The monitor program views the two ROMs installed on board locations U19 and U2Ø as ROM1 and ROM2, respectively. In other words, U19 and U2Ø are referenced as ROM1 and ROM2 by the monitor program. The relationships among the expansion ROMs, monitor program, and board locations are illustrated as follows:



Upon power-up, the monitor program will check if any system expansion ROM is installed during system reset. If the monitor program detects the presence of an auto ROM, then it will excute the program stored in the auto ROM.

In order to make it convenient for you to expand the system under monitor program version 1.1, you can place the ORG 0 assembler directive in front of the program to be stored in the auto ROM. However, under monitor program version 1.0, the auto ROM should be so designed that the starting address of ROM1 should be F000:8000 and that for ROM2 should be F000:4000.

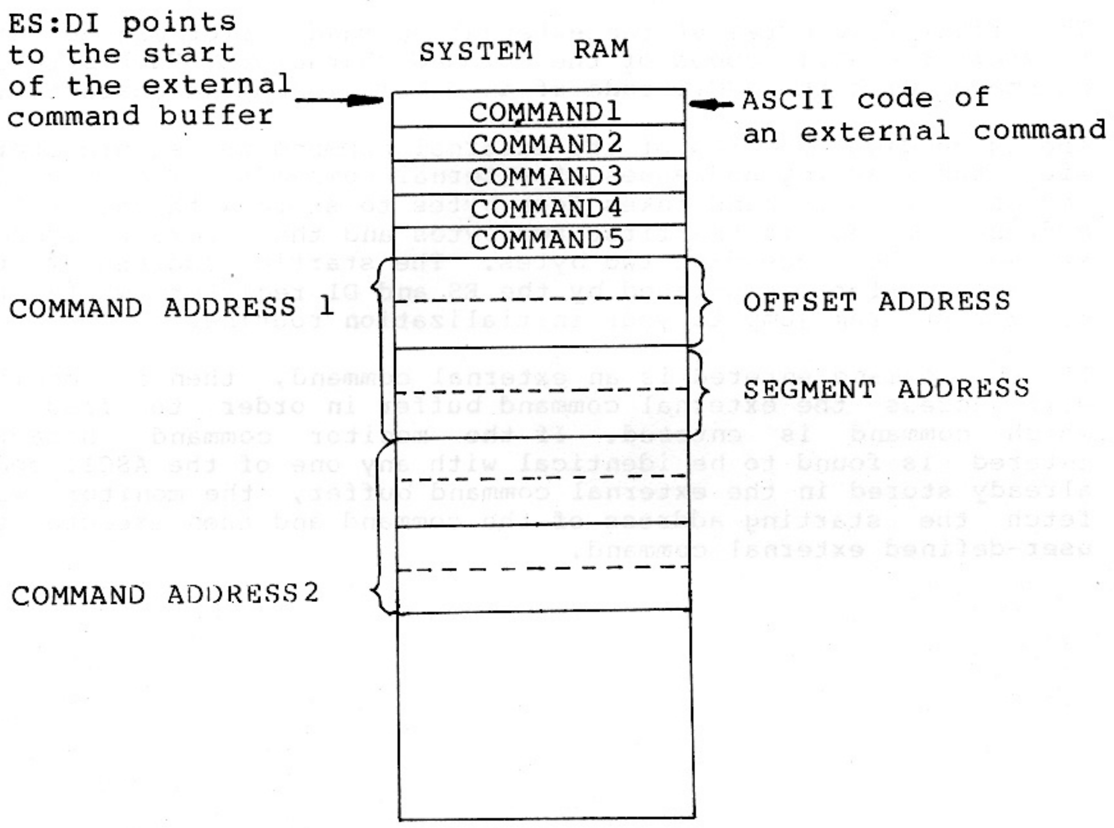
EXTERNAL COMMANDS

The monitor program version 1.1 allows user-designed monitor commands (or programs) to be executed in an ease way. The user-designed monitor commands can be (and are frequently) stored in a user-supplied EPROM. These user-designed monitor commands are referred to as external commands.

If an EPROM which contains user-defined monitor commands is installed in the system expansion ROM socket or an expansion card, and if the ROM identifier and the user-defined monitor commands are stored in the EPROM in compliance with the code arrangement rules as set forth in the chapter on MPF-I/88 system reset, the monitor program version 1.1 allows the user-defined EPROM a chance to perform an initialization operation when the existence of such an EPROM is assured (during system reset). If no chip initialization is required, the instruction RET FAR must be stored immediately after the ROM identifier. In case you desire to perform an initialization, then you should use RET FAR as the last instruction of the initialization routine. After a system reset has been performed on a system installed with user-dedfined EPROM, the monitor program "knows" that external commands in the user-defined EPROM are accessible.

The command interpreter not only processes internal (or built-in) monitor commands, it can also process external commands. Each time a command is entered, the command interpreter will check if the command is an internal command by checking a look-up table for the internal command characters. If the entered command character is not found in the internal command look-up table, the command interpreter will search the external command buffer (another area in the system RAM) in order to find out if the command entered is an external command.

The external command buffer is a memory block which is set aside by the monitor program for storing up to five external commands designed by users. The external commands are stored in the RAM as follows:

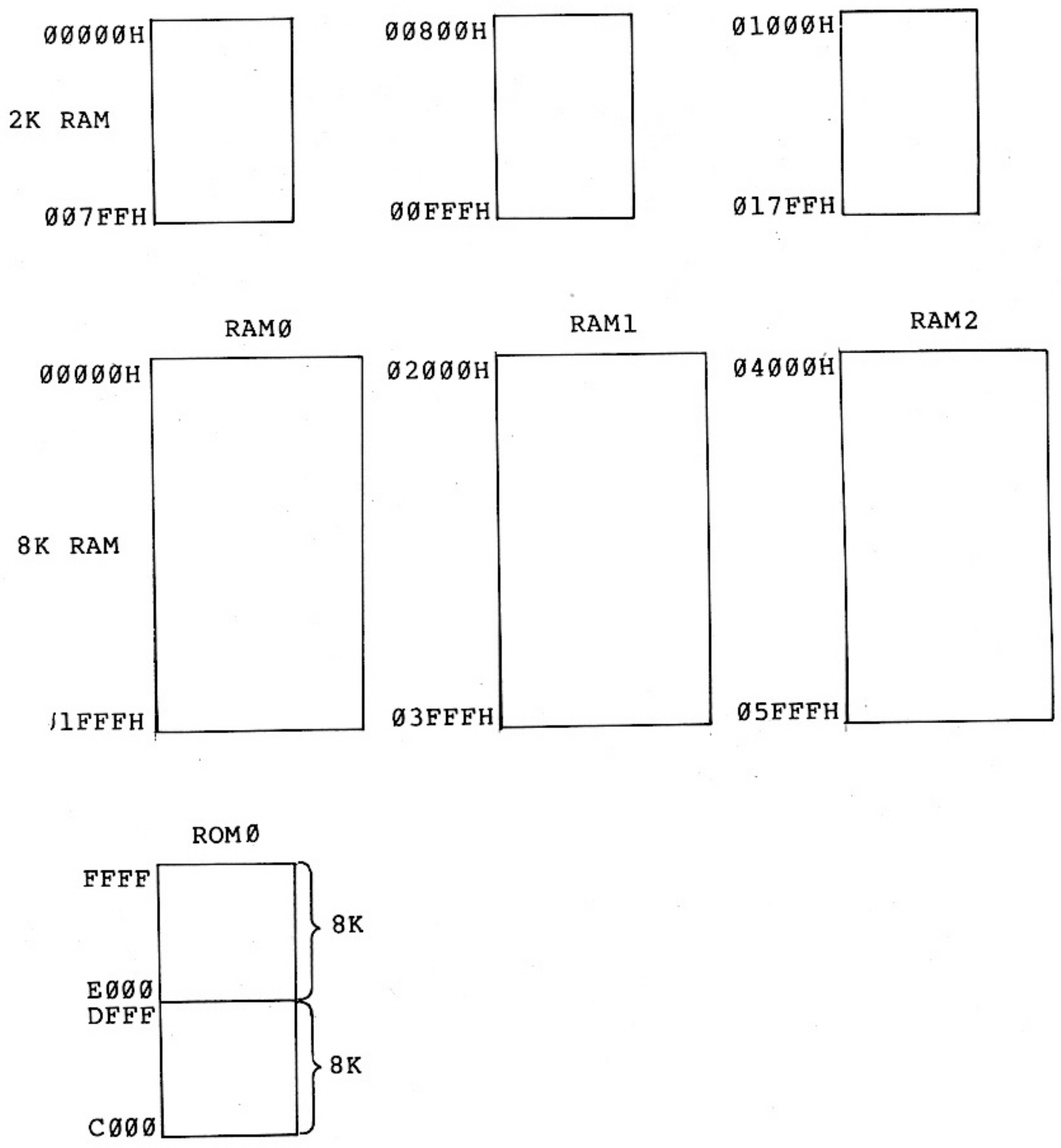

ES:DI points to the start of the external command buffer
SYSTEM RAM
COMMAND1
COMMAND2
COMMAND3
COMMAND4
COMMAND5
ASCII code of an external command
COMMAND ADDRESS 1
OFFSET ADDRESS
SEGMENT ADDRESS
COMMAND ADDRESS2

The first five bytes of the external command buffer can be used to store the ASCII codes of the command characters. Each byte can be stored with the ASCII code of an external command character.

The succeeding 2Ø bytes of the external command buffer are stored with the starting addresses of external commands. The starting address of a command takes four bytes to store with the offset address stored in the first two bytes and the segment address stored in the succeeding two bytes. The starting address of the command buffer is pointed by the ES and DI registers while the monitor program jump to your initialization routine.

If the command entered is an external command, then the monitor will access the external command buffer in order to find out which command is entered. If the monitor command character entered is found to be identical with any one of the ASCII codes already stored in the external command buffer, the monitor will fetch the starting address of the command and then execute the user-defined external command.

Appendix D Memory Map



Appendix E
I/O Port Addresses

I/O Port Addresses

The I/O port addresses assigned to I/O devices attached to the MPF-I/88 are listed as follows:

<u>LCD</u>:

	Read	Write
Command	1A2H	1AØH
Data	1A3H	1A1H

<u>Printer</u>:

Printer output data port: Port 1EØH
Printer strobe (STB): Bit 7 of port 18ØH
BUSY (printer): Bit 6 of port 1CØH.

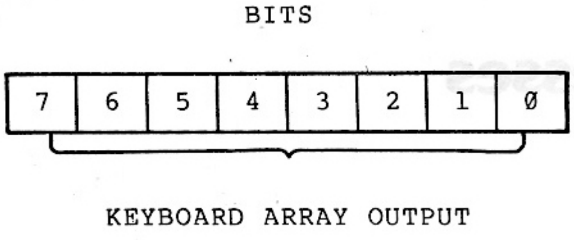
<u>Keyboard</u>:

Keyboard array output: Bit 3 through Bit Ø of port 18ØH and bit Ø through bit 7 of port 16ØH.
Keyboard array intput: Bit Ø through 4 of port 1CØH.
Control key: Bit 5 of port 1CØH.

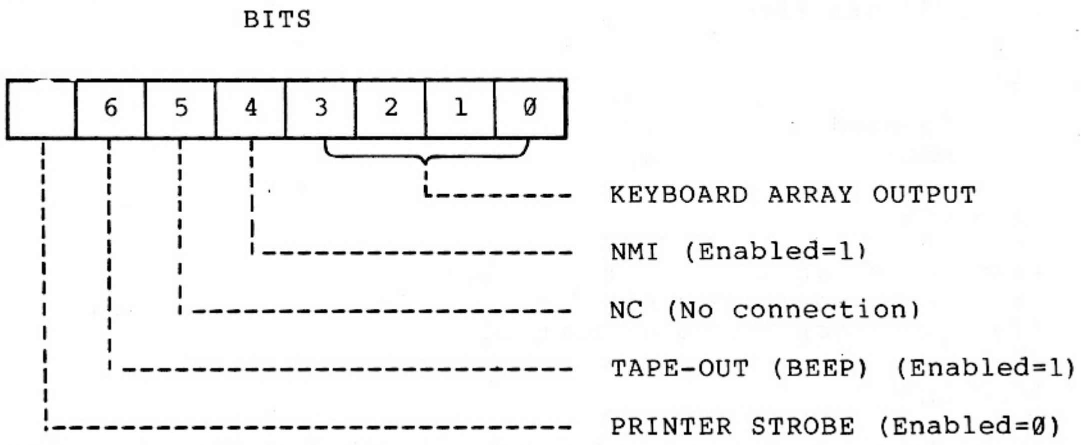
<u>TAPE-OUT (beep)</u>: Bit 6 of port 18ØH.

<u>TAPE-IN</u>: Bit 7 port 1CØH.

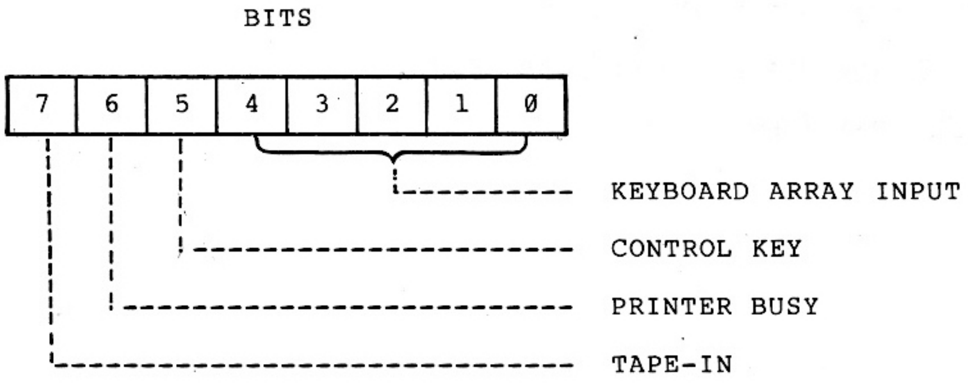
Port 16ØH:



Port 18ØH:



Port 1CØH:



Appendix F Printer Connector Pin Assignment

MPF-I/88 Printer Connector Pin Assignment

Pin	1	3	5	7	9	11	13	15
	+DØ	+D1	+D2	+D3	+D4	+D5	+D6	+D7

Pin	2	4	6	8	1Ø	12	14	16
	-STB	GND	GND	GND	GND	GND	+BUSY	N.C.

Appendix G Error Messages

1. Error Ø1

   Description: Error type 1 - SYNTAX ERROR

   This means that either your command syntax is invalid or you have a typing mistake. Make the necessary corrections.

2. Error Ø2

   Description: Error type 2 - RANGE ERROR

   This means that either the upper or lower range limits you set are incorrect or unallowable. Here are some conditions that may bring about this kind of error:

   A. Error occurs when range limits are not in the same segment. Refer to Example 1, we set the lower limit at memory location 1Ø in segment 9Ø and the upper limit at location 2Ø in segment 1ØØ. This is not allowable. The memory ranges should both be in segment 9Ø or segment 1ØØ.

      Example 1:

      ```
      >9Ø:1Ø  1ØØ:2Ø
      ```

   B. The range limits are in the same segment, however, the lower range limit is greater than the upper range limit and it is not allowable, so error occurs. Remember, the lower range limit can only be equal or smaller than the upper range limit , but never greater.

      Example 2:

      ```
      >2ØØØ,1ØØØ
      ```

   C. The address variable exceeded the maximum allowable range limits. For example, you set a segment range from 3Ø to 6Ø but you are addressing memory location 7Ø in that segment, therefore range error occurs because memory location 7Ø is not within the range of that segment.

3. Error Ø4

   Description: Error type 4 - ROM-CHECK-ERR

   Upon encountering this error, consult your local Multitech dealer.

4. Error Ø5

   Description: Error type 5 - RAM-CHECK-ERR

   Consult your local Multitech dealer when this error occurs.

5. Error Ø8

   Description: Error type 8 - TAPE-READ-ERR

   This means that the tape is defective. Try rewinding the tape and reading it again, and if still nothing happens, then something is definitely wrong with the tape.

6. Error ØE

   Description: Error type 14 - NONEXIST DEVICE

   When you get this kind of error, it may mean either one of two things. One, there is no RS232 card or two, the RS232 card is defective.

   So, whenever you see Error ØE, check if the RS232 card is installed and if the power is on. And if you still get Error ØE, then that means the RS232 card is defective, please consult your local Multitech dealer.

Appendix H
Table of ASCII Codes

The following is the table of ASCII codes supported by MPF-I/88.

*** Table of ASCII Codes ***

MSD		0	1	2	3	4	5	6	7
LSD		0000	0001	0010	0011	0100	0101	0110	0111
0	0000			space	0	@	P	'	p
1	0001			!	1	A	Q	a	q
2	0010			″	2	B	R	b	r
3	0011			#	3	C	S	c	s
4	0100			$	4	D	T	d	t
5	0101			%	5	E	U	e	u
6	0110			&	6	F	V	f	v
7	0111	Bell		/	7	G	W	g	w
8	1000	← (BS)		(	8	H	X	h	x
9	1001	TAB ↔		)	9	I	Y	i	y
A	1010	LF		*	:	J	Z	j	z
B	1011		ESC	+	;	K	[	k	{
C	1100	FF		,	<	L	|	l	/
D	1101	CR		—	=	M	]	m	}
E	1110			•	>	N		n	~
F	1111			/	?	O		o	

Note 1:

The ASCII codes for some special function keys such as F1, F2, SHIFT-F1, and SHIFT-F2 are not listed in the table. Their ASCII codes are:

F1: 81H   F2: 82H   SHIFT-F1: 83H   SHIFT-F2: 84H

Note also that the ASCII codes for these special function keys are never displayed.

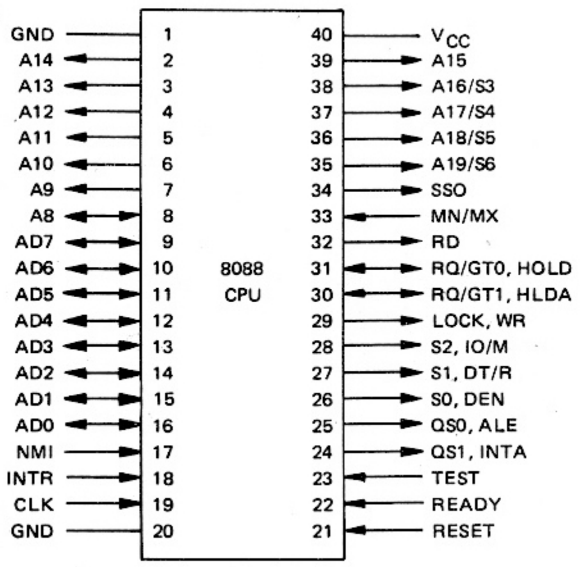
Note 2:

The CAPLOCK key does not have an ASCII code. A position code is used to detect whether this key is pressed.

Note 3:

If a key, say S, is pressed while holding down the ALT key, then the key code so generated is the sum of the ASCII code of that key (S) and 80H (hexadecimal).

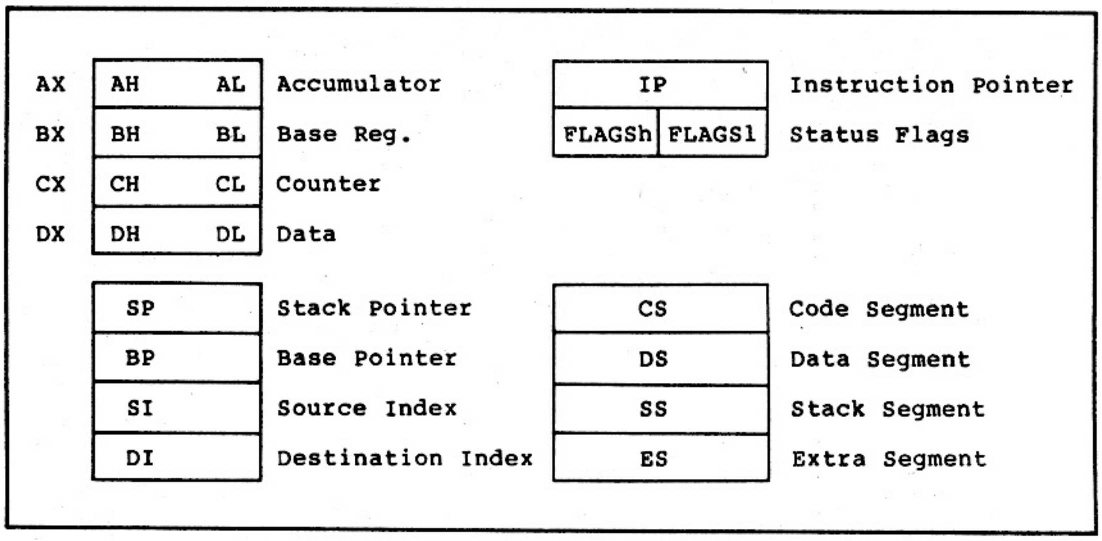
Appendix I
CPU Pin Assignment



Pin Name	Description	Type
AD0-AD7	Address/Data Bus	Bidirectional, tristate
A8-A15	Address Bus	Output, tristate
A16/S3, A17/S4	Address/Segment identifier	Output, trisate
A18/S5	Address/Interrupt enable status	Output, tristate
A19/S6	Address/status	Output, tristate
SSO	Status output	Output, tristate
RD	Read control	Output, tristate
READY	Wait state request	Input
TEST	Wait for test control	Input
INTR	Interrupt request	Input
NIMI	Non-maskable interrupt request	Input
RESET	System Reset	Input
CLK	System Clock	Input
MN/MX	= GND for a maximum system	
S0, S1, S2	Machine cycle status	Output, tristate
RQ/GT0, RQ/GT1	Local bus priority control	Bidirectional
QS0, QS1	Instruction queue status	Output
LOCK	Bus hold control	Output, tristate
MN/MX	= V_{CC} for a minimum system	
IO/M	Memory or I/O access	Output, tristate
WR	Write control	Output, tristate
ALE	Address Latch enable	Output
DT/R	Data transmit/receive	Output, tristate
DEN	Data enable	Output, tristate
INTA	Interrupt acknowledge	Output, tristate
HOLD	Hold request	Input
HLDA	Hold acknowledge	Output
V_{CC}, GND	Power, ground	
Maximum System Signals		Minimum System Signals

8088 Pins and Signal Assignments

Appendix J 8088 Registers



Appendix K
8086/8088 Instruction Set Summary

This summary is presented only for reference use. For those instructions that can have a variety of operands, the different possibilities are listed. Please refer to outside documentation for more detailed explanations of the exact use of these instructions.

Data Transfer Instructions

A. MOV - Move:

- register or memory location to or from register
- immediate to register or memory location
- immediate to register
- memory location to accumulator
- accumulator to memory location
- register or memory location to segment register
- segment register to register or memory location

B. PUSH - Push:

- register or memory location
- register
- segment register

C. POP - Pop:

- register or memory location
- register
- segment register

D. XCHG - Exchange:

- register or memory location with register
- register with register

E. IN - Input from:

- fixed port
- variable port

F. OUT - Output to:

- fixed port
- variable port

G. XLAT - Translate byte to AL:

H. LEA - Load EA (Effective Address) to register

I. LDS - Load pointer value to DS and register

J. LES - Load pointer value to ES and register

K. LAHF - Load AH with flags

L. SAHF - Store AH into flags

M. PUSHF - Push flags

N. POPF - Pop flags

Arithmetic Instructions

A. ADD - Add:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

B. ADC - Add with carry:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

C. INC - Increment:

- register or memory location
- register

D. AAA - ASCII adjust for addition

E. DAA - Decimal adjust for addition

F. SUB - Subtract:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

G. SBB - Subtract with borrow:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

H. DEC - Decrement:

- register or memory location
- register

I. NEG - Negate the contents of a specified register or memory location

J. CMP - Compare:

- register or memory location with register
- immediate with register or memory location
- immediate with accumulator

K. AAS - ASCII adjust for subtract

L. DAS - Decimal adjust for subtract

M. MUL - Multiply (unsigned) accumulator by register or memory location

N. IMUL - Integer multiply (signed) accumulator by register memory location

O. AAM - ASCII adjust for multiplication

P. DIV - Divide (unsigned) accumulator by register or memory location

Q. IDIV - Integer divide (signed) accumulator by register or memory location

R. AAD - ASCII adjust for division

S. CBW - Convert byte to word and perform sign extension from AL to AX

T. CWD - Convert word to doubleword and perform sign extension from AX to DX

Logical Instructions

A. NOT - Ones complement of register or memory location

B. SHL - Logical left shift of register

C. SAL - Arithmetic left shift of register

D. SHR - Logical right shift of register

E. SAR - Arithmetic right shift of register

F. ROL - Rotate register left

G. ROR - Rotate register right

H. RCL - Rotate register left through carry flag

I. RCR - Rotate register right through carry flag

J. AND - Logical and:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

K. TEST - Logical AND test with result stored in flags

- register or memory location with register
- immediate data and register or memory location
- immediate data and accumulator

L. OR - Logical or:

- register or memory location with register with result stored in either

- immediate to register or memory location
- immediate to accumulator

M. XOR - Exclusive or:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

String Manipulation Instructions

A. REP - Repeat

B. MOVS - Move byte or word

C. CMPS - Compare byte or word

D. SCAS - Scan byte or word

E. LODS - Load byte or word to AL or AX

F. STDS - Store byte or word from AL or AX

Control Transfer Instructions

A. CALL - Call subroutine:

- direct within segment
- indirect within segment
- direct intersegment
- indirect intersegment

B. JMP - Unconditional jump:

- direct within segment
- direct within segment-short
- indirect within segment
- direct intersegment
- indirect intersegment

C. RET - Return from CALL:

- within segment
- within segment and add immediate to stack pointer
- intersegment
- intersegment and add immediate to stack pointer

D. JE or JZ - Jump on equal or zero **

** **The jump instructions that are listed in pairs are exactly identical and can be used interchangably.**

E. JL or JNGE - Jump on less or not greater or equal

F. JLE or JNG - Jump on less or equal or not greater

G. JB or JNAE - Jump on below or not above or equal

H. JBE or JNA - Jump on below or equal or not above

I. JP or JPE - Jump on parity or parity even

J. JO - Jump on overflow

K. JS - Jump on sign

L. JNE or JNZ - Jump on not equal or not zero

M. JNL or JGE - Jump on not less or greater or equal

N. JNLE or JG - Jump on not less or equal or greater

O. JNB or JAE - Jump on not below or above or equal

P. JNBE or JA - Jump on not below or equal or above

Q. JNP or JPO - Jump on not parity or parity odd

R. JNO - Jump on not overflow

S. JNS - Jump on not sign

T. JCXZ - Jump on CX zero

U. LOOP - Subtract one from CX and jump if greater than zero

V. LOOPZ or LOOPE - Loop while zero or equal

W. LOOPNZ or LOOPNE - Loop while not zero or equal

X. INT - Interrupt

 - Type specified

Y. INTO - Interrupt on overflow

Z. IRET - Return from interrupt

Processor Control Instructions

A. CLC - Clear carry

B. CMC - Complement carry

C. STC - Set carry

D. CLD - Clear direction

E. STD - Set direction

F. CLI - Clear interrupt

G. STI - Set interrupt

H. HLT - Halt

I. WAIT - Wait

J. ESC - Escape to external device

K. LOCK - Bus lock prefix

Appendix L
8086/8088 Instruction Set Listed Alphabetically

Instruction		Object Code	Bytes	Clock Periods
AAA		37	1	4
AAD		D5 0A	2	60
AAM		D4 0A	2	83
AAS		3F	1	4
ADC	ac,data	0001010w kk [jj]	2 or 3	4
ADC	mem/reg$_1$,data	100000sw mod 010 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
ADC	mem/reg$_1$,mem/reg$_2$	000100dw mod rrr r/m [DISP] [DISP]	2, 3 or 4 2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
ADD	ac,data	0000010w kk [jj]	2 or 4	4
ADD	mem/reg,data	100000sw mod 000 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
ADD	mem/reg$_1$,mem/reg$_2$	000000dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg. to mem: 16 + EA
AND	ac,data	0010010w kk [jj]	2 or 4	4
AND	mem/reg,data	1000000w mod 100 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
AND	mem/reg$_1$,mem/reg$_2$	001000dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
CALL	addr	9A kk jj hh gg	5	28
CALL	disp 16	E8 kk jj	3	19
CALL	mem	FF mod 011 r/m [DISP] [DISP]	2, 3 or 4	32-bit mem pointer: 37 + EA
CALL	mem/reg	FF mod 010 r/m [DISP] [DISP]	2, 3 or 4	16-bit reg pointer: 16 16 bit mem pointer: 21 + EA

Instruction		Object Code	Byte	Clock Periods
CBW		98	1	2
CLC		F8	1	2
CLD		FC	1	2
CLI		FA	1	2
CMC		F5	1	2
CMP	ac,data	0011110w kk [jj]	2 or 3	4
CMP	mem/reg,data	100000sw mod 111 r/m [DISP] [DISP] kk [jj]	3, or 4 or 6	reg: 4 mem: 10 + EA
CMP	mem/reg$_1$,mem/reg$_2$	001110dw mod rrr r/m [DISP] [DISP]	2,3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 9 + EA
CMPS		1010011w	1	22 9 + 22/repetition*
CWD		99	1	5
DAA		27	1	4
DAS		2F	1	4
DEC	mem/reg	1111111w mod 001 r/m [DISP] [DISP]	2,3 or 4	reg: 3 mem: 15 + EA
DEC	16-bit reg	01001 rrr	1	2
DIV	mem/reg	1111011w mod 110 r/m [DISP] [DISP]	2,3 or 4	8-bit reg: 80 → 90 16-bit reg: 144 → 162 8-bit mem: (86 → 96) + EA 16-bit mem: (150 → 168) + EA
ESC	mem/reg	11011xxx mod xxx r/m [DISP] [DISP]	2,3 or 4	mem: 8 + EA reg: 2
HLT		F4	1	2
IDIV	mem/reg	1111011w mod 111 r/m [DISP] [DISP]	2,3 or 4	8-bit reg: 101 → 112 16-bit reg: 165 → 184 8-bit mem: (107 → 118) + EA 16-bit mem: (171 → 190) + EA
IMUL	mem/reg	1111011w mod 101 r/m [DISP] [DISP]	2,3 or 4	8-bit reg: 80 → 98 16-bit reg: 128 → 154 8-bit mem: (86 → 104) + EA 16-bit mem: (134 → 160) + EA
IN	ac, DX	1110110w	1	8
IN	ac, port	1110010w	2	10

Instruction		Object Code	Bytes	Clock Periods
INC	mem/reg	1111111w	2, 3 or 4	reg: 3
		mod 000 r/m		mem: 15 + EA
		[DISP]		
		[DISP]		
INC	16-bit reg	01000rrr	1	2
INT		11001100*	1	52
		11001101	2	51
		type		
INTO		CE	1	interrupt: 53
				no interrupt: 4
IRET		CF	1	24
JA	disp	77	2	4/No Branch
JNBE		disp		16/Branch
JAE	disp	73	2	4/No Branch
JNB		disp		16/Branch
JB	disp	72	2	4/No Branch
JNAE		disp		8/Branch
JBE	disp	76	2	4/No Branch
JNA		disp		16/Branch
JCXZ	disp	E3	2	6/No Branch
		disp		18/Branch
JE	disp	74	2	4/No Branch
JZ		disp		16/Branch
JG	disp	7F	2	4/No Branch
JNLE		disp		16/Branch
JGE	disp	7D	2	4/No Branch
JNL		disp		16/Branch
JL	disp	7C	2	4/No Branch
JNGE		disp		16/Branch
JLE	disp	7E	2	4/No Branch
JNG		disp		16/Branch
JMP	addr	EA	5	15
		kk		
		jj		
		hh		
		gg		
JMP	disp	EB	2	15
		disp		
JMP	disp 16	E9	3	15
		kk		
		jj		
JMP	mem	FF	2, 3 or 4	mem ptr 32:
		mod 101 r/m		24 + EA
		[DISP]		
		[DISP]		
JMP	mem/reg	FF	2, 3 or 4	reg ptr 16:
		mod 100 rr/m		11
		[DISP]		mem ptr 16:
		[DISP]		18 + EA
JNE	disp	75	2	4/No Branch
JNZ		disp		16/Branch
JNO	disp	71	2	4/No Branch
		disp		16/Branch
JNP	disp	7B	2	4/No Branch
JPO		disp		16/Branch
JNS	disp	79	2	4/No Branch
		disp		16/Branch
JO	disp	70	2	4/No Branch
		disp		16/Branch

* Implied type = 3

Instruction	Object Code	Bytes	Clock Periods
JP disp	7A	2	4/No Branch
JPE	disp		16/Branch
JS disp	78	2	4/No Branch
	disp		16/Branch
LAHF	9F	1	4
LDS reg,mem	C5	2, 3 or 4	16 + EA
	mod rrr r/m		
	[DISP]		
	[DISP]		
LEA reg,mem	8D	2, 3 or 4	2 + EA
	mod rrr r/m		
	[DISP]		
	[DISP]		
LES reg,mem	C4	2, 3 or 4	16 + EA
	mod rrr r/m		
	[DISP]		
	[DISP]		
LOCK	F0	1	2
LODS	1010110w	1	12
			9 + 13/repetition*
LOOP disp	E2	2	5/No Branch
	disp		17/Branch
LOOPE disp	E1	2	6/No Branch
LOOPZ	disp		18/Branch
LOOPNE disp	E0	2	5/No Branch
LOOPNZ	disp		19/Branch
MOV mem/reg$_1$,mem/reg$_2$	100010dw	2, 3 or 4	reg to reg: 2
	mod rrr r/m		reg to mem: 8 + EA
	[DISP]		mem to reg: 9 + EA
	[DISP]		
MOV reg,data	1011 wrrr	2 or 3	4
	kk		
	[jj]		
MOV ac,mem	1010000w	3	10
	kk		
	jj		
MOV mem,ac	1010001w	3	10
	kk		
	jj		
MOV segreg,mem/reg	8E	2, 3 or 4	reg to reg: 2
	mod 0rr r/m		mem to reg: 8 + EA
	[DISP]		
	[DISP]		
MOV mem/reg,segreg	8C	2, 3 or 4	reg to reg: 2
	mod 0rr r/m		reg to mem: 9 + EA
	[DISP]		
	[DISP]		
MOV mem/reg,data	1100011w	3, 4, 5 or	reg/mem: 10 + EA
	mod 000 r/m	6	
	[DISP]		
	[DISP]		
	kk		
	[jj]		
MOVS	1010010w	1	18
			9 + 17/repetition*

* When preceded by REP prefix

Instruction		Object Code	Bytes	Clock Periods
MUL	mem/reg	1111011w mod 100 r/m [DISP] [DISP]	2, 3 or 4	8-bit reg: 70 → 77 16-bit reg: 118 → 133 8-bit mem: (76 → 83) + EA 16-bit mem: (124 → 139) + EA
NEG	mem/reg	1111011w mod 011 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 mem: 16 + EA
NOP		90	1	3
NOT	mem/reg	1111011w mod 010 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 mem: 16 + EA
OR	ac,data	0000110w kk [jj]	2 or 3	4
OR	mem/reg,data	1000000w mod 001 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
OR	mem/reg$_1$,mem/reg$_2$	000010dw mod rrr r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
OUT	DX,ac	1110111w	1	8
OUT	port,ac	1110011w yy	2	10
POP	mem/reg	8F mod 000 r/m [DISP] [DISP]	2, 3 or 4	reg: 8 mem: 17 + EA
POP	reg	01011rrr	1	8
POP	segreg	000ss111	1	8
POPF		9D	1	8
PUSH	mem/reg	FF mod 110 r/m [DISP] [DISP]	2, 3 or 4	reg: 11 mem: 16 + EA
PUSH	reg	01010rrr	1	10
PUSH	segreg	000ss110	1	10
PUSHF		9C	1	10
RCL	mem/reg,count	110100cw mod 010 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)

N = count value in CL

Instruction		Object Code	Bytes	Clock Periods
RCR	mem/reg,count	110100cw mod 011 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
REP	/REPE/REPNE	1111001z	1	2
RET	(Inter-segment)	CB	1	18
RET	(Intra-segment)	C3	1	8
RET	disp 16 (Inter-segment)	CA kk jj	3	17
RET	disp 16 (Intra-segment)	C2 kk jj	3	12
ROL	mem/reg,count	110100cw mod 000 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
ROR	mem/reg,count	110100cw mod 001 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
SAHF		9E	1	4
SAR	mem/reg,count	110100cw mod 111 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
SBB	ac,data	0001110w kk [jj]	2 or 3	4
SBB	mem/reg,data	100000sw mod 011 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 rem: 17 + EA
SBB	mem/reg$_1$,mem/reg$_2$	000110dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg from reg: 3 mem from reg: 9 + EA reg from mem: 16 + EA
SCAS		1010111w	1	15 9 + 15/repetition*
SEG	segreg	001ss110	1	2
SHL SAL	mem/reg,count	110100cw mod 100 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem: 15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)

* When preceded by REP prefix
N = count value in CL

Instruction	Object Code	Bytes	Clock Periods
SHR mem/reg,count	110100cw	2, 3 or 4	count = 1
	mod 101 r/m		reg: 2
	[DISP]		mem: 15 + EA
	[DISP]		count = [CL]
			reg: 8 + (4 * N)
			mem: 20 + EA + (4 * N)
STC	F9	1	2
STD	FD	1	2
STI	FB	1	2
STOS	1010101w	1	11
			9 + 10/repetition*
SUB ac,data	0010110w	2 or 3	4
	kk		
	[jj]		
SUB mem/reg,data	100000sw	3, 4, 5 or	reg: 4
	mod 101 r/m	6	mem: 17 + EA
	[DISP]		
	[DISP]		
	kk		
	[jj]		
SUB mem/reg$_1$,mem/reg$_2$	001010dw	2, 3 or 4	reg from reg: 3
	mod rrr r/m		mem from reg: 9 + EA
	[DISP]		reg from mem: 16 + EA
	[DISP]		
TEST ac,data	1010100w	2 or 3	4
	kk		
	[jj]		
TEST mem/reg,data	1111011w	3, 4, 5 or	reg: 5
	mod 000 r/m	6	mem: 11 + EA
	[DISP]		
	[DISP]		
	kk		
	[jj]		
TEST reg,mem/reg	1000010w	2, 3 or 4	reg with reg: 3
	mod rrr r/m		reg with mem: 9 + EA
	[DISP]		
	[DISP]		
WAIT	9B	1	3(min.) + 5n
XCHG reg,ac	10010rrr	1	3
XCHG reg,mem/reg	100011w	2, 3 or 4	reg with reg: 4
	mod rrr r/m		reg with mem: 17 + EA
	[DISP]		
	[DISP]		
XLAT	D7	1	11
XOR ac,data	0011010w	2 or 3	4
	kk		
	[jj]		
XOR mem/reg,data	100000w	3, 4, 5 or	reg: 4
	mod 110 r/m	6	mem: 17 + EA
	[DISP]		
	[DISP]		
	kk		
	[jj]		
XOR mem/reg$_1$,mem/reg$_2$	001100dw	2, 3 or 4	reg with reg: 3
	mod rrr r/m		mem with reg: 9 + EA
	[DISP]		reg with mem: 16 + EA
	[DISP]		

* When preceded by REP prefix
n = clocks per sample of the TEST input

Appendix M 8086/8088 Instruction Set Object Codes in Numeric Sequence Ascending

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
00	mod reg r/m	[disp] [disp]	ADD mem/reg,reg (byte)
01	mod reg r/m	[disp] [disp]	ADD mem/reg,reg (word)
02	mod reg r/m	[disp] [disp]	ADD reg. mem/reg (byte)
03	mod reg r/m	[disp] [disp]	ADD reg, mem/reg (word)
04	kk		ADD AL,kk
05	kk	jj	ADD AX, jjkk
06			PUSH ES
07			POP ES
08	mod reg r/m	[disp] [disp]	OR mem/reg,reg (byte)
09	mod reg r/m	[disp] [disp]	OR mem/reg,reg (word)
0A	mod reg r/m	[disp] [disp]	OR reg,mem/reg (byte)
0B	mod reg r/m	[disp] [disp]	OR reg,meme/reg (word)
0C	kk		OR AL,kk
0D	kk	jj	OR AL,jjkk
0E			PUSH CS
0F			Not used
10	mod reg r/m	[disp] [disp]	ADC meme/reg,reg (byte)
11	mod reg r/m	[disp] [disp]	ADC meme/reg,reg (word)
12	mod reg r/m	[disp] [disp]	ADC reg,mem/reg (byte)
13	mod reg r/m	[disp] [disp]	ADC reg,mem/reg (word)
14	kk		ADC AL,kk
15	kk	jj	ADC AX,jjkk
16			PUSH SS
17			POP SS
18	mod reg r/m	[disp] [disp]	SBB mem/reg,reg (byte)
19	mod reg r/m	[disp] [disp]	SBB mem/reg,reg (word)
1A	mod reg r/m	[disp] [disp]	SBB reg,mem/reg (byte)
1B	mod reg r/m	[disp] [disp]	SBB reg,mem/reg (word)
1C	kk		SBB AL,kk
1D	kk	jj	SBB AX,jjkk
1E			PUSH DS
1F			POP DS
20	mod reg r/m	[disp] [disp]	AND mem/reg,reg (byte)
21	mod reg r/m	[disp] [disp]	AND mem/reg,reg (word)
22	mod reg r/m	[disp] [disp]	AND reg,mem/reg (byte)
23	mod reg r/m	[disp] [disp]	AND reg,mem/reg (word)
24	kk		AND AL,kk
25	kk	jj	AND AX,jjkk
26			SEG ES
27			DAA
28	mod reg r/m	[disp] [disp]	SUB mem/reg,reg (byte)
29	mod reg r/m	[disp] [disp]	SUB mem/reg,reg (word)
2A	mod reg r/m	[disp] [disp]	SUB reg,mem/reg (byte)
2B	mod reg r/m	[disp] [disp]	SUB reg,mem/reg (word)
2C	kk		SUB AL,kk
2D	kk	jj	SUB AX,jjkk
2E			SEG CS
2F			DAS

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
30	mod reg r/m	[disp] [disp]	XOR mem/reg,reg (byte)
31	mod reg r/m	[disp] [disp]	XOR mem/reg,reg (word)
32	mod reg r/m	[disp] [disp]	XOR reg,mem/reg (byte)
33	mod reg r/m	[disp] [disp]	XOR reg,mem/reg (word)
34	kk		XOR AL,kk
35	kk	jj	XOR AX,jjkk
36			SEG SS
37			AAA
38	mod reg r/m	[disp] [disp]	CMP mem/reg,reg (byte)
39	mod reg r/m	[disp] [disp]	CMP mem/reg,reg (word)
3A	mod reg r/m	[disp] [disp]	CMP reg,mem/reg (byte)
3B	mod reg r/m	[disp] [disp]	CMP reg,mem/reg (word)
3C	kk		CMP AL,kk
3D	kk	jj	CMP AX,jjkk
3E			SEG DS
3F			AAS
40			INC AX
41			INC CX
42			INC DX
43			INC BX
44			INC SP
45			INC BP
46			INC SI
47			INC DI
48			DEC AX
49			DEC CX
4A			DEC DX
4B			DEC BX
4C			DEC SP
4D			DEC BP
4E			DEC SI
4F			DEC DI
50			PUSH AX
51			PUSH CX
52			PUSH DX
53			PUSH BX
54			PUSH SP
55			PUSH BP
56			PUSH SI
57			PUSH DI
58			POP AX
59			POP CX
5A			POP DX
5B			POP BX
5C			POP SP
5D			POP BP
5E			POP SI
5F			POP DI
60-6F			Not Used

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
70	disp		JO disp
71	disp		JNO disp
72	disp		JB or JNAE or JC disp
73	disp		JNB or JAE or JNC disp
74	disp		JE or JZ disp
75	disp		JNE or JNZ disp
76	disp		JBE or JNA disp
77	disp		JNBE or JA disp
78	disp		JS disp
79	disp		JNS disp
7A	disp		JP or JPE disp
7B	disp		JNP or JPO disp
7C	disp		JL or JNGE disp
7D	disp		JNL or JGE disp
7E	disp		JLE or JNG disp
7F	disp		JNLE or JG disp
80	mod 000 r/m	[disp] [disp] kk	ADD mem/reg,kk
80	mod 001 r/m	[disp] [disp] kk	OR mem/reg,kk
80	mod 010 r/m	[disp] [disp] kk	ADC mem/reg,kk
80	mod 011 r/m	[disp] [disp] kk	SBB mem/reg,kk
80	mod 100 r/m	[disp] [disp] kk	AND mem/reg,kk
80	mod 101 r/m	[disp] [disp] kk	SUB mem/reg,kk
80	mod 110 r/m	[disp] [disp] kk	XOR mem/reg,kk
80	mod 111 r/m	[disp] [disp] kk	CMP mem/reg,kk
81	mod 000 r/m	[disp] [disp] kkjj	ADD mem/reg,jjkk
81	mod 001 r/m	[disp] [disp] kkjj	OR mem/reg,jjkk
81	mod 010 r/m	[disp] [disp] kkjj	ADC mem/reg,jjkk
81	mod 011 r/m	[disp] [disp] kkjj	SBB mem/reg,jjkk
81	mod 100 r/m	[disp] [disp] kkjj	AND mem/reg,jjkk
81	mod 101 r/m	[disp] [disp] kkjj	SUB mem/reg,jjkk
81	mod 110 r/m	[disp] [disp] kkjj	XOR mem/reg,jjkk
81	mod 111 r/m	[disp] [disp] kkjj	CMP mem/reg,jjkk
82	mod 000 r/m	[disp] [disp] kk	ADD mem/reg,kk (byte)
82	xx 001 xxx		Not used
82	mod 010 r/m	[disp] [disp] kk	ADC mem/reg,kk (byte)
82	mod 011 r/m	[disp] [disp] kk	SBB mem/reg,kk (byte)
82	xx 100 xxx		Not used
82	mod 101 r/m	[disp] [disp] kk	SUB mem/reg,kk (byte)
82	xx 110 xxx		Not used
82	mod 111 r/m	[disp] [disp] kk	CMP mem/reg,kk (byte)
83	mod 000 r/m	[disp] [disp] kk	ADD mem/reg,jjkk (word-sign extended)
83	xx 001 xxx		Not used
83	mod 010 r/m	[disp] [disp] kk	ADC mem/reg,jjkk (word-sign extended)
83	mod 011 r/m	[disp] [disp] kk	SBB mem/reg,jjkk (word-sign extended)
83	xx 100 r/m		Not used
83	mod 101 r/m	[disp] [disp] kk	SUB mem/reg,jjkk (word-sign extended)
83	xx 110 xxx		Not used
83	mod 111 r/m	[disp] [disp] kk	CMP mem/reg,jjkk (word-sign extended)
84	mod reg r/m	[disp] [disp]	TEST mem/reg,reg (byte)
85	mod reg r/m	[disp] [disp]	TEST mem/reg,reg (word)
86	mod reg r/m	[disp] [disp]	XCHG reg,mem/reg (byte)
87	mod reg r/m	[disp] [disp]	XCHG reg,mem/reg (word)
88	mod reg r/m	[disp] [disp]	MOV mem/reg,reg (byte)
89	mod reg r/m	[disp] [disp]	MOV mem/reg,reg (word)

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
8A	mod reg r/m	[disp] [disp]	MOV reg,mem/reg (byte)
8B	mod reg r/m	[disp] [disp]	MOV reg,mem/reg (word)
8C	mod 0ss r/m	[disp] [disp]	MOV mem/reg,segreg
8C	x 1 xxxxx		Not used
8D	mod reg r/m	[disp] [disp]	LEA reg,addr
8E	mod 0ss r/m	[disp] [disp]	MOV segreg, mem/reg
8E	xx 1 xxxxx		Not used
8F	mod 000 r/m	[disp] [disp]	POP mem/reg
8F	xx 001 xxx		Not used
8F	xx 010 xxx		Not used
8F	xx 011 xxx		Not used
8F	xx 100 xxx		Not used
8F	xx 101 xxx		Not used
8F	xx 110 xxx		Not used
8F	xx 111 xxx		Not used
			Not used
90			NOP
91			XCHG AX,CX
92			XCHG AX,DX
93			XCHG AX,BX
94			XCHG AX,SP
95			XCHG AX,BP
96			XCHG AX,SI
97			XCHG AX,DI
98			CBW
99			CWD
9A	kk	jj hh gg	CALL addr
9B			WAIT
9C			PUSHF
9D			POPF
9E			SAHF
9F			LAHF
A0	qq	pp	MOV AL,addr
A1	qq	pp	MOV AX,addr
A2	qq	pp	MOV addr,AL
A3	qq	pp	MOV addr,AX
A4			MOVS BYTE
A5			MOVS WORD
A6			CMPS BYTE
A7			CMPS WORD
A8	kk		TEST, AL,kk
A9	kk	jj	TEST AX,jjkk
AA			STOS BYTE
AB			STOS WORD
AC			LODS BYTE
AD			LODS WORD
AE			SCAS BYTE
AF			SCAS WORD

Object Code			Mnemonic
Byte #0	Byte #1	Succeeding Bytes	
B0	kk		MOV AL,kk
B1	kk		MOV CL,kk
B2	kk		MOV DL,kk
B3	kk		MOV BL,kk
B4	kk		MOV AH,kk
B5	kk		MOV CH,kk
B6	kk		MOV DH,kk
B7	kk		MOV BH,kk
B8	kk	jj	MOV AX,jjkk
B9	kk	jj	MOV CX,jjkk
BA	kk	jj	MOV DX,jjkk
BB	kk	jj	MOV BX,jjkk
BC	kk	jj	MOV SP,jjkk
BD	kk	jj	MOV BP,jjkk
BE	kk	jj	MOV SI,jjkk
BF	kk	jj	MOV DI,jjkk
C0			Not used
C1			Not used
C2	kk	jj	RET jjkk
C3			RET
C4	mod reg r/m	[disp] [disp]	LES reg,addr
C5	mod reg r/m	[disp] [disp]	LDS reg,addr
C6	mod 000 r/m	[disp] [disp] kk	MOV mem,kk
C6	xx 001 xxx		Not used
C6	xx 010 xxx		Not used
C6	xx 011 xxx		Not used
C6	xx 100 xxx		Not used
C6	xx 101 xxx		Not used
C6	xx 110 xxx		Not used
C6	xx 111 xxx		Not used
C7	mod 000 r/m	[disp] [disp] kkjj	MOV mem,jjkk
C7	xx 001 xxx		Not used
C7	xx 010 xxx		Not used
C7	xx 011 xxx		Not used
C7	xx 100 xxx		Not used
C7	xx 101 xxx		Not used
C7	xx 110 xxx		Not used
C7	xx 111 xxx		Not used
C8			Not used
C9			Not used
CA	kk	jj	RET jjkk
CB			RET
CC			INT 3
CD	type		INT Type
CE			INTO
CF			IRET

Object Code			Mnemonic
Byte #0	Byte #1	Succeeding Bytes	
D0	mod 000 r/m	[disp] [disp]	ROL mem/reg,1 (byte)
D0	mod 001 r/m	[disp] [disp]	ROR mem/reg,1 (byte)
D0	mod 010 r/m	[disp] [disp]	RCL mem/reg,1 (byte)
D0	mod 011 r/m	[disp] [disp]	RCR mem/reg,1 (byte)
D0	mod 100 r/m	[disp] [disp]	SAL or SHL mem/reg,1 (byte)
D0	mod 101 r/m	[disp] [disp]	SHR mem/reg,1 (byte)
D0	xx 110 xxx		Not used
D0	mod 111 r/m	[disp] [disp]	SAR mem/reg,1 (byte)
D1	mod 000 r/m	[disp] [disp]	ROL mem/reg,1 (word)
D1	mod 001 r/m	[disp] [disp]	ROR mem/reg,1 (word)
D1	mod 010 r/m	[disp] [disp]	RCL mem/reg,1 (word)
D1	mod 011 r/m	[disp] [disp]	RCR mem/reg,1 (word)
D1	mod 100 r/m	[disp] [disp]	SAL or SHL mem/reg,1 (word)
D1	mod 101 r/m	[disp] [disp]	SHR mem/reg,1 (word)
D1	xx 110 xxx		Not used
D1	mod 111 r/m	[disp] [disp]	SAR mem/reg,1 (word)
D2	mod 000 r/m	[disp] [disp]	ROL mem/reg,CL (byte)
D2	mod 001 r/m	[disp] [disp]	ROR mem/reg,CL (byte)
D2	mod 010 r/m	[disp] [disp]	RCL mem/reg,CL (byte)
D2	mod 011 r/m	[disp] [disp]	RCR mem/reg,CL (byte)
D2	mod 100 r/m	[disp] [disp]	SAL or SHL mem/reg,CL (byte)
D2	mod 101 r/m	[disp] [disp]	SHR mem/reg,CL (byte)
D2	xx 110 xxx		Not used
D2	mod 111 r/m	[disp] [disp]	SAR mem/reg,CL (byte)
D3	mod 000 r/m	[disp] [disp]	ROL mem/reg,CL (word)
D3	mod 001 r/m	[disp] [disp]	ROR mem/reg,CL (word)
D3	mod 010 r/m	[disp] [disp]	RCL mem/reg,CL (word)
D3	mod 011 r/m	[disp] [disp]	RCR mem/reg,CL (word)
D3	mod 100 r/m	[disp] [disp]	SAL or SHL mem/reg,CL (word)
D3	mod 101 r/m	[disp] [disp]	SHR mem/reg,CL (word)
D3	xx 110 xxx		Not used
D3	mod 111 r/m	[disp] [disp]	SAR mem/reg,CL (word)
D4	0A		AAM
D5	0A		AAD
D6			Not used
D7			XLAT
D8	mod xxx r/m	[disp] [disp]	ESC mem/reg
D9	mod xxx r/m	[disp] [disp]	ESC mem/reg
DA	mod xxx r/m	[disp] [disp]	ESC mem/reg
DB	mod xxx r/m	[disp] [disp]	ESC mem/reg
DC	mod xxx r/m	[disp] [disp]	ESC mem/reg
DD	mod xxx r/m	[disp] [disp]	ESC mem/reg
DE	mod xxx r/m	[disp] [disp]	ESC mem/reg
DF	mod xxx r/m	[disp] [disp]	ESC mem/reg
E0	disp		LOOPNE/LOOPNZ disp
E1	disp		LOOPE/LOOPZ disp
E2	disp		LOOP disp
E3	disp		JCXZ disp
E4	kk		IN AL,kk
E5	kk		IN AX,kk
E6	kk		OUT kk,AL
E7	kk		OUT kk,AX
E8	disp	disp	CALL disp 16
E9	disp	disp	JMP disp 16

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
EA	kk	jj hh gg	JMP addr
EB	disp		JMP disp
EC			IN AL,DX
ED			IN AX,DX
EE			OUT DX,AL
EF			OUT DX,AX
F0			LOCK
F1			Not used
F2			REPNE or REPNZ
F3			REP or REPE or REPZ
F4			HLT
F5			CMC
F6	mod 000 r/m	[disp] [disp] kk	TEST mem/reg,kk
F6	xx 001 xxx		Not used
F6	mod 010 r/m	[disp] [disp]	NOT mem/reg (byte)
F6	mod 011 r/m	[disp] [disp]	NEG mem/reg (byte)
F6	mod 100 r/m	[disp] [disp]	MUL mem/reg (byte)
F6	mod 101 r/m	[disp] [disp]	IMUL mem/reg (byte)
F6	mod 110 r/m	[disp] [disp]	DIV mem/reg (byte)
F6	mod 111 r/m	[disp] [disp]	IDIV mem/reg (byte)
F7	mod 000 r/m	[disp] [disp] kkjj	TEST mem/reg,jjkk
F7	xx 001 xxx		Not used
F7	mod 010 r/m	[disp] [disp]	NOT mem/reg (word)
F7	mod 011 r/m	[disp] [disp]	NEG mem/reg (word)
F7	mod 100 r/m	[disp] [disp]	MUL mem/reg (word)
F7	mod 101 r/m	[disp] [disp]	IMUL mem/reg (word)
F7	mod 110 r/m	[disp] [disp]	DIV mem/reg (word)
F7	mod 111 r/m	[disp] [disp]	IDIV mem/reg (word)
F8			CLC
F9			STC
FA			CLI
FB			STI
FC			CLD
FD			STD
FE	mod 000 r/m	[disp] [disp]	INC mem/reg (byte)
FE	mod 001 r/m	[disp] [disp]	DEC mem/reg (byte)
FE	xx 010 xxx		Not used
FE	xx 011 xxx		Not used
FE	xx 100 xxx		Not used
FE	xx 101 xxx		Not used
FE	xx 110 xxx		Not used
FE	xx 111 xxx		Not used
FF	mod 000 r/m	[disp] [disp]	INC mem/reg (word)
FF	mod 001 r/m	[disp] [disp]	DEC mem/reg (word)
FF	mod 010 r/m	[disp] [disp]	CALL mem/reg
FF	mod 011 r/m	[disp] [disp]	CALL mem
FF	mod 100 r/m	[disp] [disp]	JMP mem/reg
FF	mod 101 r/m	[disp] [disp]	JMP mem
FF	mod 110 r/m	[disp] [disp]	PUSH mem
FF	xx 111 xxx		Not used

Multitech
INDUSTRIAL CORP

OFFICE/
15FL. 135 CHIEN KUO N. ROAD, SEC. 2, TAIPEI
10479, TAIWAN, R.O.C.
TEL: (02)505-5533
TELEX: 19162 MULTIIC FAX: (02)505-4451
FACTORY/
1 INDUSTRYE, ROAD, III,
HSINCHU SCIENCE-BASED INDUSTRIAL PARK
HSINCHU, TAIWAN 300, R.O.C.

DOC. NO.: M8801-8605D