MPF-I/88

Reference Manual

MPF-I/88

Reference Manual

Table of Contents

Preface

The MPF-I/88 is designed as a teaching aid for you to practising 8088 assembly language programming. With the MPF-I/88, you can write 8088 assembly language programs and test the programs. You can even develop your own microcomputer system based on the MPF-I/88.

The MPF-I/88 can be used by a programmer, who is already familiar with basic computer concepts and assembly language programming yet intends to learn programming a 16-bit microprocessor such as 8088. However, this learning kit can also be used by a beginner, who has no computer background and has never used a microcomputer before, to learn some basic computer concepts and assembly language programming.

As you read this manual, it is assumed that you have finished reading MPF-I/88 User's Manual and has run the sample program presented in that manual.

The MPF-I/88 Reference Manual provides the information you need to understand the internal operations in more detail, and thus, to use the system more flexibly.

Chapter 1 describes how to use the interrupt service routines supported by the monitor program of the system.

Chapter 2 gives a closer look of the operations performed during a cold or warm system reset.

Chapter 3 introduces input/output programming concepts.

Chapter 4 gives a circuit description of the MPF-I/88.

Chapter 5 describes how i/o device drivers for the MPF-I/88 were designed and their functions. This chapter guides you to read the the MPF-I/88 Monitor Program Source Listing and teaches you some program tracing techniques. You need to refer to the Monitor Program Source Listing and some example programs while reading this chapter.

Chapter 5 also introduces to you some of the frequently used Macro Assembler directives supported by MS-DOS Macro Assembler of Microsoft, which was used for the development of the MPF-I/88 monitor program.

Appendix A is designed for those who are not familiar with the 8088 assembly language instruction set. Explanation is provided for each individual instruction. Some simple but useful examples are given so that you may get a quick lesson of the instruction set and how each instruction is used. For the beginners who is not familiar with the 8088 instruction set, it would be better to begin with Appendix A. Although that appendix is valuable, you should not rely totally on Appendix A as a comprehensive hard-ware/software tutorial. You need to refer to other documentations in order to get a thorough understanding of the 8088 microprocessor.

Appendix D provides a list of the books which should be referenced as you learn to program the 8088 microprocessor.

Chapter 1

How to Use Interrupt Subroutines

A set of useful interrupt subroutines are built in the MPF-I/88 monitor program. Each of these subroutines performs a pre-defined function such as returning control to the monitor program, inputting a character from the current console, generating a beep sound, or outputting a character to the console, etc. You can refer to an individual chapter in the MPF-I/88 User's Guide for a detailed description of the functions performed by these useful subroutines.

Sometimes you may wish to perform a specific subroutine function within your program. In this case, there is no need to write all the instructions comprising the interrupt service subroutine. You can simply use an INT instruction in your program to invoke the desired subroutine.

To use the interrupt service subroutines, you must first read the chapter on useful subroutines in the MPF-I/88 User's Guide. Some subroutines calls for the user to supply a value (it is sometimes referred to as input parameter) to the appropriate register or registers, while others require no input parameters. After the selected subroutine is executed, some subroutines will return a value to the appropriate register(s). The contents of some registers will be affected after the execution of some interrupt subroutine. All of such information is described in detail in that chapter.

If the interrupt service subroutine you intend to use requires that input parameters be loaded into the appropriate register(s), then you have to load the register(s) to be used by the interrupt sevice subroutine accordingly prior to using the INT instruction to invoke the service subroutine. When the execution of an interrupt service subroutine affects the contents of register(s), you should save the value of the registers whose value is to be affected by the execution of the interrupt service subroutine before using the INT instruction.

Chapter 2

MPF-I/88 System Reset

The following is a brief description of the tasks performed during system reset. The MPF-I/88 performs a cold reset (cold start) when power is turned on. A warm start is performed when the RESET key is pressed.

During a cold start, the system performs the following tasks:

1. Display the sign-on message "MPF-I/88", and the version number of the monitor program.
2. Perform a RAM test.
3. Perform a ROM checksum test.

COLD START

When power is first applied to the system, the CPU will begin executing the monitor program starting from physical address FFFFØH. This 2Ø-bit actual address is calculated by adding the segment address FFFFH and the effective address ØH in the following way.

```
    FFFFX  ---> Code segment address
 +   ØØØØ  ---> Effective address
 ---------
    FFFFØ  ---> Actual address
```

Note that the segment address is first shifted left four bits for calculating the actual address. While the segment address is left-shifted, zeroes are shifted into the four least significant bits to form a 2Ø-bit segment address.

Since only 16 bytes of memory are available between memory locations FFFFØH and FFFFFH (not enough for a large program), a jump-to-FCØØ3H instruction is executed as the first instruction of the monitor program. Then the monitor program will determine whether a cold start or a warm start is to be performed.

RAM Test

In case a cold start is to be performed, the RAM test will be performed first. The RAM test routine will write the two word patterns "5555" and "AAAA" into each memory word and read back the contents. If the contents of the memory word read match what was written into that memory word, the RAM check routine will continue to check the next word. If a mismatch is found, then the RAM may contain bad storage cells and the routine will display an error message.

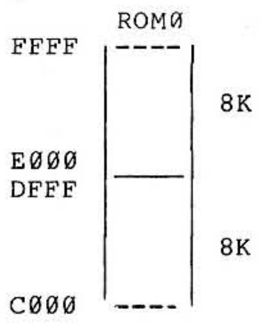
Note that the RAM test routine checks contiguous memory space. If the system RAM is not configured to reside in contiguous memory space, then only the low order memory range will be checked.

ROM Checksum Test

For the ROM checksum test, the contents of memory words are added together to form a checksum which is stored in a memory word. If the value of the low order byte of the memory word is zero, then we assume the ROM is tested O.K. Otherwise, an error message will be displayed.

The ROM checksum routine works in a much more complicated manner than the RAM test routine. The complexity of the software design of the ROM checksum routine is due to considerations aiming at making the system flexible for future system expansion. Before describing the programming logic of the ROM checksum routine, we will describe the possible ROM mappings.

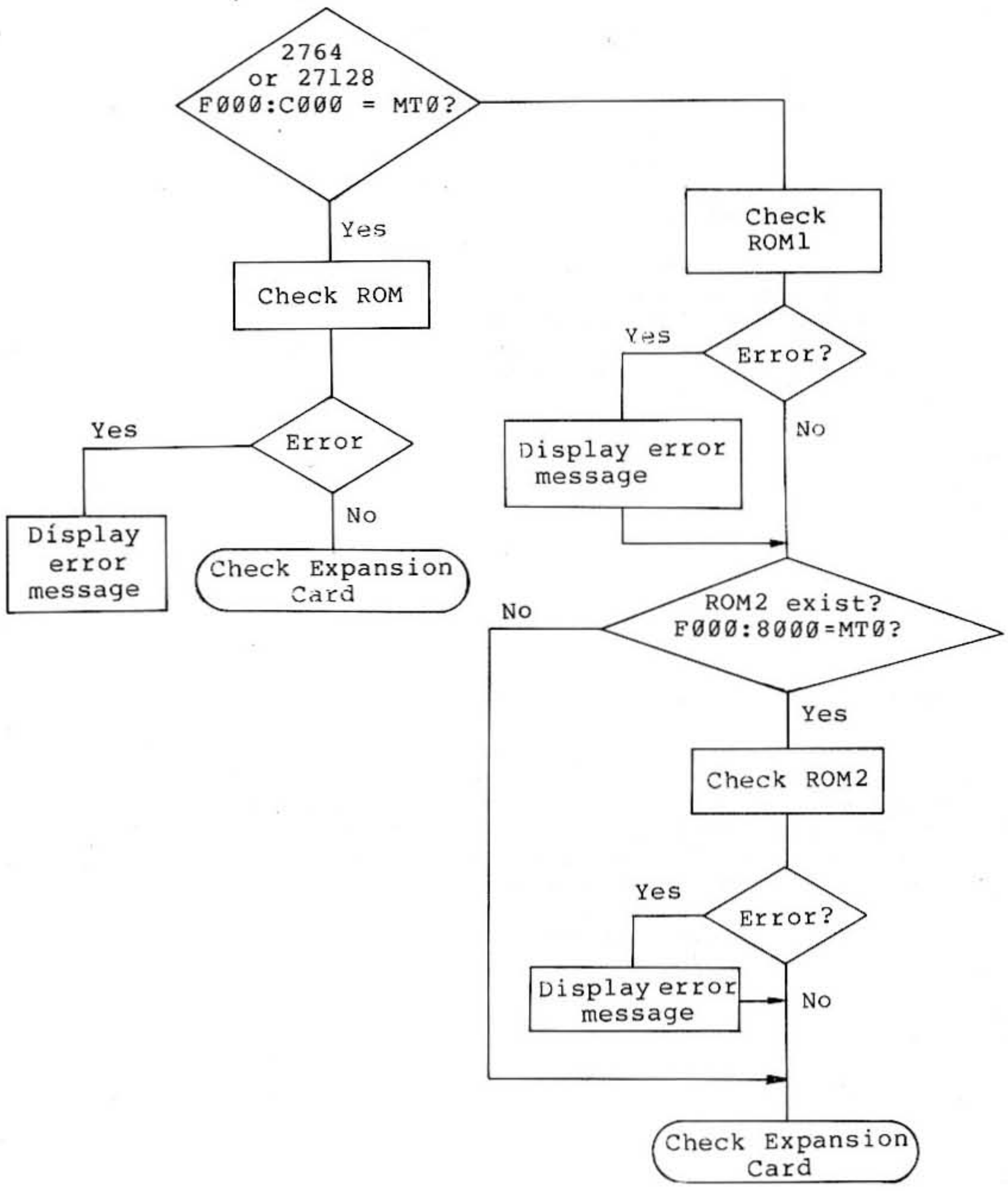
The standard MPF-I/88 is built with one 27128 with a 16K memory space. The memory space in such a configuration is illustrated as follows. ROMØ is the one with memory address starting from CØØØH through FFFFH. The segment address assigned to ROMØ is FØØØH.



*** ROM Memory Map ***

Although one 27128 is used as ROM chip on the system, the ROM checksum routine treats the system in such a way that two 2764s can also be inserted as ROMs on the system board.

A flowchart is illustrated as follows for the ROM checksum routine.


2764
or 27128
F000:C000 = MT0?
Yes
Check ROM
Error
Yes
No
Display error message
Check Expansion Card
Check ROM1
Error?
Yes
No
Display error message
ROM2 exist?
F000:8000=MT0?
No
Yes
Check ROM2
Error?
Yes
Display error message
No
Check Expansion Card

The ROM checksum routine starts by checking if the ROM chips on the system board are 2764 ROM chips or 27128 ROM chips. A three-byte ROM identifier is stored in the first three bytes of each ROM. The first two bytes are stored with the two characters -- MT -- which represents Multitech. When the ROM checksum routine detects the two characters, it determines that the ROM is stored with the codes designed by Multitech. The third byte of a Multitech's ROM is always filled with an ASCII characters in the range from Ø through 6. The ROM checksum routine decodes the three-byte ROM idenfifier as follows:

```
MTØ = ROM2 (in which MPF-I/88 line assembler and dis-
      assembler are stored.)
MT1 = TVB - TV interface ROM
MT2 = Auto-run ROM such as a BASIC interpreter.
MT3 = Printer interface ROM
MT4 = ROM for EPROM programmer board
```

The ROM checksum routine distinguishes a 2764 and 27128 by checking if the contents of the first three bytes starting from CØØØH are MTØ. If it is, then 27128 is used as the monitor ROM. Otherwise, the routine assumes that 2764s are used as the monitor ROM.

If a 27128 is detected, the routine will proceed to perform the actual ROM testing procedure. If an error is detected, it will display the error message. Otherwise, it will proceed to perform the expansion card test.

If a 2764 is detected, the routine will proceed to perform the actual ROM testing procedure by checking ROM1 first. If an error is detected, it will display the error message and proceed to detect whether ROM2 exists. It determines whether ROM2 is inserted by checking whether the contents of the first three bytes starting from 8ØØØH are MTØ. If it is, ROM2 exists. Otherwise, it will proceed to perform the expansion card test. If an error is detected during ROM2 testing, it will display the error message and then proceed to perform the expansion card test.

If an error is detected during a RAM or ROM test, you are suggested to replace the defective RAM or ROM chip with a good one. If you don't know how to replace the ICs, consult your local distributor for service.

Expansion Card Test

Expansion cards are assigned the segment address EØØØH. The expansion card test routine tests memory in 4K bytes increments. The first three bytes of the ROM module on an expansion card are stored with ROM identifier as mentioned before.

When an expansion card is detected, the initialization routine for that card will be executed. The address of the initialization routine is stored starting from the fourth byte on the expansion card. However, if an auto-run ROM is detected on the expansion card, the code stored in that auto-run ROM will be executed immediately. Thus, no initialization routine is executed when an auto-run ROM is detected.

Since the initialization routine starts with a CALL FAR instruction, you must use a RET FAR instruction starting from the fourth byte on the expansion card to skip the initialization routine if you intend to design your own applications with an initialization routine. However, when designing your own applications, be careful not to change the contents of the system stack.

If auto-run ROMs are inserted on both the system board and the expansion card, the one on the expansion card is executed since it is assigned with higher priority than the auto-run ROM on the system board.

A cold reset cycle is completed when the expansion card test is finished.

WARM START

During a warm start, the system performs the following tasks:

1. Create system vector table.
2. Create interrupt vector table.
3. Initialize printer port.
4. Initialize keyboard.
5. Initialize RS-232 port - Baud rate 9600, two stop bit, even parity check, seven-bit word length.
6. Generate a beep sound.
7. Check if an AUTO-RUN ROM is inserted in the empty socket reserved for an AUTO-RUN ROM or on an expansion card. If an AUTO-RUN ROM is present, the program contained in that AUTO-RUN ROM executes automatically.

To return the control to the monitor program, you can use the interrupt instruction INT 7.

Chapter 3

I/O Programming

This chapter is a brief introduction to input/output programming on the MPF-I/88. The information provided here allows you to gain some ideas on I/O device programming. The chapter on useful monitor subroutines gives you some information on using built-in service routines to perform a variety of I/O tasks, but curious users and those developing special applications may be interested in writing their own routines to perform I/O directly.

If you do not intend to program the the I/O devices directly, this chapter can be skipped without harm. But as you get more and more familiar with the hardware and software of 8Ø88, you may want to refer to this chapter.

If you are interested in I/O programming, you should not rely totally on this chapter to get familiar with I/O programming concepts. The monitor program source listing and the information presented in the Software Reference Manual are also very valuable sources of information.

There is a wide variety of I/O devices on the MPF-I/88. Some of the I/O devices such as the screen and the printer can be used for output only. Others such as the keyboard are capable of inputting data. Others, such as the tape interfaces or serial interface, can both input and output data.

For an I/O device to be functional, the three following conditions must be met:

1) An interface circuit must be available. An interface circuit is a communication line via which the I/O device can talk to the system.

2) An I/O device driver must be available. A device driver is a program which drives the I/O device so that the I/O device can interact with the system.

3) The I/O device must be installed or loaded into system memory.

If you intend to design your own I/O devices in the future, you may need to write your own device drivers. You may also include a device driver in your own application program for a specific application. The best way to learn I/O programming is to trace the existing device driver programs instruction by instruction. For the MPF-I/88 system, the device drivers are all included in the Monitor Program Source Listing.

3.1 I/O PORTS

3.1.1 Memory-Mapped I/O

There are two common ways to design I/O support circuitry on microcomputers. One is that I/O devices may be memory-mapped. I/O devices may be accessed through memory locations. In a computer system, an I/O device is said to be memory-mapped if it is accessed through memory locations.

3.1.2 I/O-Mapped I/O

A more common practice in computer design is to use I/O ports for data transfer between the system and external devices. Each I/O device configured in a system is assigned with a specific I/O port address. The MPF-I/88 uses this latter method. When data is to be transferred to a device, the OUT instruction is used. When data is to be transferred from a device to the system, the IN instruction is used.

3.2 I/O Port Addresses

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

LCD:

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

Printer:

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

Keyboard:

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.

TAPE-OUT (beep): Bit 6 of port 18ØH.

TAPE-IN: Bit 7 port 1CØH.

3.3 The Printer Driver

The printer driver is a routine which is designed to send data from the system to the printer. Data to be output to the printer is first sent to the printer port 1EØH through AL register. The system will then test if the printer is busy by checking bit 6 of port 1CØH. If the printer is not busy, data will then be sent to printer buffer. When a low (zero) is sensed on the $\overline{\text{STROBE}}$ line (bit 7 of 18ØH), data is sent from the printer buffer to the printer.

When the printer routine is called, the system will first save the system status by pushing the contents of all registers onto the stack and then alter the contents of the Data Segment register so as to point to system data. After a data transfer has been completed between the system and the printer, the $\overline{\text{STROBE}}$ line will again be pulled high and the printer driver will return the result of a data transfer via the AH register. If a data transfer is performed successfully, then a zero will be returned to the AH register. If a data transfer is not performed successfully, then a one will be returned to the AH register.

3.4 Programming the Display

The LCD is programmed through four I/O ports - 1AØH, 1A1H, 1A2H, and 1A3H. When a command is to be written to the LCD, port 1AØH is used. When a command read is to be performed, port 1A2H is used. Data is output to the LCD via port 1A1H, and input to the LCD via port 1A3H.

To program the LCD, you must refer to the data sheet of the LCD display, which is in an appendix, and the display driver, which is included in the MPF-I/88 Monitor Program Source Listing. The programming techniques for LCD is also explained in a chapter of the MPF-I/88 Reference Manual.

3.5 Keyboard

The keyboard matrix consists of 12 column lines and five row lines. The system scans the keyboard every 15 milliseconds. During a keyboard scan one of the 12 column lines connecting to the bit Ø through bit 3 of port 18ØH and 8 bits of I/O port 16ØH and is pulled low, while the other 11 column lines are high. When a key is pressed, a low pulse is sensed on one of the row lines and a code is sent to the system. Please refer to the keyboard matrix chart and a chapter of the MPF-I/88 Software Reference Manual.

3.5.1 The Control Key

Bit 5 of output port 1CØH is used by the Control key. When the Control key on the keyboard is pressed, this pin is active.

3.6 Audio Interface

When data is to be transferred from the system to tape, it is sent through bit 6 of output port 18ØH using the OUT instruction. Output to the buzzer is also sent through this bit.

When data is to be read from tape to system, it is sent through bit 7 of input port 1CØH using the IN instruction.

3.6.1 Tape Format

Each time the system writes data to tape, data is recorded onto tape in a fixed tape format. The tape format is defined below:

1) Leader tone: 256 consecutive bytes of ØØ.

2) Sync bit: 1

3) Sync byte: Ø16H

4) Data: 256 bytes of data are stored as a block (data record).

5) CRC bytes: Each data block is followed by two CRC (Cyclic Redundancy Check) bytes.

6) Tailer : The tailer consists of four bytes.

The leader tone is designed to act as a signal, enabling the system or tape recorder to detect incoming data when data is to be transferred.

The sync (short for synchronization) bit is sometimes referred to as a framing bit or start-stop bit. The system outputs this bit to tell an external device (in this case the tape recorder) that a data transmission is to occur.

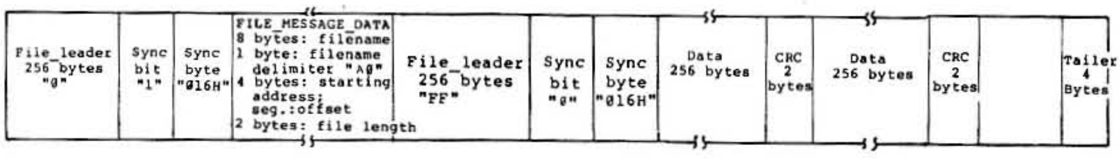
The sync byte is used as a data transmission protocol; i.e., when data is read from tape to system, if a correct sync byte is read, then data can be transmitted to the system. If the correct sync byte is not found, the interface driver will search for the leader again.

Data is stored on tape as a series of 256 bytes of data records. Each 256-byte data record is followed by two CRC bytes. The CRC bytes are used for checking errors during data transmission. The CRC bytes are written onto tape after each data record when data is stored onto tape. When data is read from tape to system, the system will generate two CRC bytes according to the preceding data record read. Then the two CRC bytes so generated will be compared with the CRC bytes. If the contents of CRC bytes match, this signals that a data record is transmitted correctly. Otherwise, an error occurred during the data transmission.

The tailer marks the end of a file.

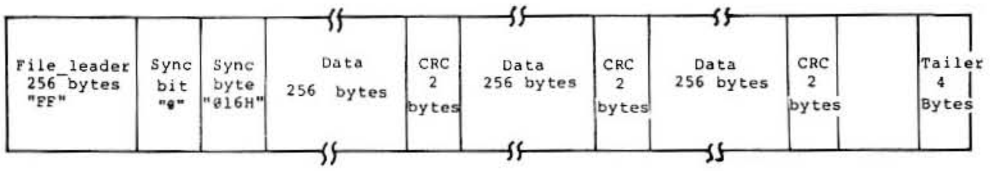
The MPF-I/88 tape format is illustrated as follows:

MPF I/88 TAPE FORMAT



The cassette interface driver also allows cassette tape on which data is stored in IBM PC cassette tape format to be loaded into system memory. The IBM tape format is illustrated as follows:

IBM PC TAPE FORMAT



1) Leader: 256 consecutive bytes of FF .

2) Sync bit: Ø

3) Sync byte: Ø16H

4) Filename: The filename is stored in eight bytes.

5) Filename delimiter: One byte of filename delimiter - ØØ - is stored immediately following a filename.

6) Starting address: Four bytes are used for storing the starting address of file. The starting address consists of two bytes for the segment address and two bytes for the offset address.

5) File length : The length of a file is stored in two bytes.

The data record on an IBM formatted tape is also formed by 256 bytes, which is followed by two CRC bytes.

Chapter 4

MPF-I/88 Circuit Description

This chapter will give you a brief circuit description of the MPF-I/88. After reading this chapter, you will have some ideas on how the hardware components function in the system. For the readers who are not interested in the hardware aspects of the system, this chapter can be skipped. However, for the readers who are interested in the hardware and intend to expand the system for their own special applications, this chapter should be read thoroughly while tracing the schematics.

We will cover the functional components of the system in the following order:

1. The CPU and its support circuitry, including
   1) System timing circuit,
   2) System wait logic,
   3) System reset circuit,
   4) Interrupt logic,
   5) Bus buffer,
   6) Memory and I/O device decoders.

2. System memory,

3. Input/Output interface logic.

THE CPU AND ITS SUPPORTING LOGICS

1) System Timing Circuit

The system timing circuit consists of a 14.318 MHz crystal oscillator and the 74LS04 at board location U4. A frequency of 14.318 MHz is generated at pin 2 of U4. This signal is divided by three to obtain a frequency of 4.77 MHz at pin 3 after going through the divide-by-three circuit at U10. The clock frequency of 4.77 MHz is supplied to pin 19 of U3 (the CPU) as system clock and the 62-pin expansion slot (EXT-BUS).

2) Wait Logic

The wait state logic is necessary to pull the READY input (pin 22) of 8088 low while the system is performing an I/O read or write. The wait state logic consists of the ICs on U5, U6, U9, U13, and U14. When the system (CPU) is going to perform an I/O read or write, the outputs of 8088's $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 29) will be ANDed at the AND gate at U6, whose output will then be sent to pin 1 of the dual 2 to 4 line decoder at U14. Since the 8088's IO/$\overline{M}$ output (pin 28) is sent to pin 2 of U14, pin 4, 5, 6, 7 of U14 will generate one of such signals as $\overline{MEMR}$, $\overline{IOR}$, $\overline{MEMW}$, $\overline{IOW}$ depending on the states of the two inputs of pin 1 and pin 2. The signal lines of $\overline{MEMR}$, $\overline{IOR}$, $\overline{MEMW}$, $\overline{IOW}$ are connected to the pins B12, B14, B11, and B13 of EXT-BUS through the octal bus driver 74LS244 at U13.

$\overline{IOR}$ and $\overline{IOW}$, after going through EXT-BUS, will again be ANDed at U6 and output from pin 3 of U6 to pin 13 of U5 (a quad 2 input OR gate). The output of the OR gate will be supplied to the READY input of 8088 (pin 22) through a dual D type flip flop 74LS74 at U9 in order to generate a wait cycle. (Note that the input to pin 11 of U9 is supplied by pin 2 of U10 so that the negative portion of a system clock cycle can be used for system synchronization.) In case a longer period of wait state is needed by a peripheral device, a low pulse can be sent through line A10 I/O CHANNEL READY of the EXT-BUS to pin 13 of U9 to insert extra wait states.

3) System Reset Circuit

The RESET signal is first sent to the system reset circuit at pin 1 of U8 (74LS14 - a hex inverter Schmitt Trigger) from the keyboard. After the RESET signal is squared up by the Schmitt Trigger, it is supplied through pin 3 (of the Schmitt Trigger) to pin 21 of the CPU to initialize a system reset cycle. The RESET signal is also present to line B2 of the EXT-BUS. Pin 4 of the Schmitt Trigger is connected to pin 1 of U21 and U22 (octal D type flip flop with clear) to clear the contents of the flip flop.

4) Interrupt Logic

An interrupt request generated by a peripheral device is first sent to the CPU through line B8 of EXT-BUS to pin 13 of U8. It is then sent to INTR (pin 18 of 8088) through pin 11 of U4 (a hex inverter). After the INTR signal is accepted by the 8088, it will generate an interrupt acknowledge (INTA), a low active pulse. The low pulse is sent to the interrupting device through line B5 of the EXT-BUS.

Users can apply a shorting plug (close jumper) at JP2 in order to route interrupt requests from IRQ2, IRQ3, IRQ4, and IRQ7 to the 8088. We will describe how to route interrupt requests to the 8088 later. Note the close jumper is provided in a standard MPF-I/88 package. It is illuatrated as follows:

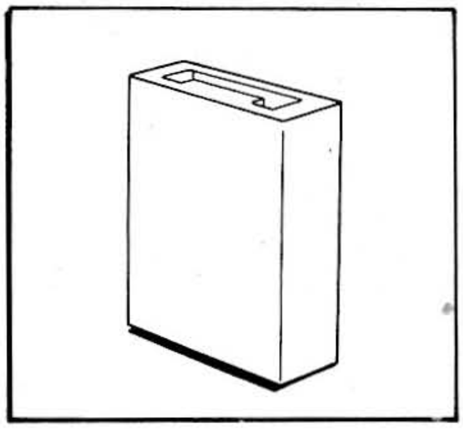
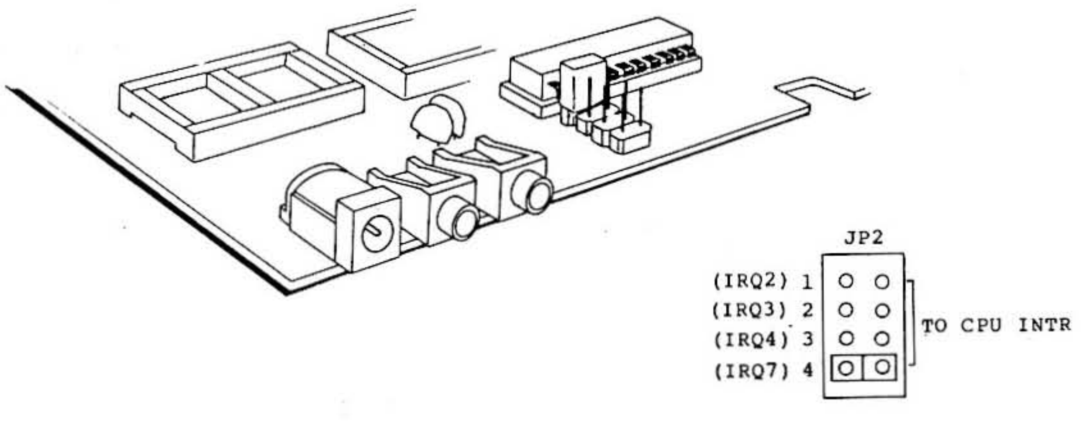


Diagram of a Close Jumper

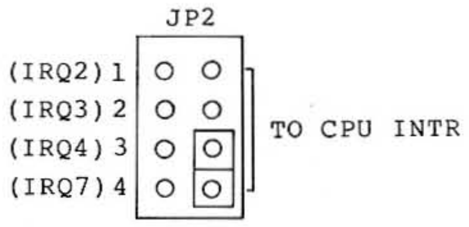
If you have an adapter card such as the IBM Parallel Printer Adapter (which uses IRQ7 to interrupt the 8Ø88), you can plug this adapter card into the optional expansion unit (or a 62-pin H connector which you can solder to the position reserved for it on the main PC board) and apply the shorting plug to route IRQ7 to 8Ø88. Note that the shorting plug is applied to the desired pair of pins as illustrated in the following chart.



The hardware design of MPF-I/88 allows the 8259 Programmable Interrupt Controller to be used to handle interrupt processing. If you intend to use an 8259 to handle interrupt processing with the system, the shorting plug is applied to JP2 in a different way.

Take for example that an adapter card using IRQ2 to interrupt the 8088 is to be used together with the 8259 interrupt controller. You can plug this adapter card into the optional expansion unit (or an on-board 62-pin H connector installed on the main PC board by yourself). And then plug the 8259 card into the expansion unit.

After you have configured the system this way, the close jumper can be applied to JP2 as follows. Note that when an 8259 is used in the system, the close jumper should only be applied to any two of the four pins on the right column, and 8259 pin 17 (INT) is connected to pin B8 (INTR) of the expansion unit.



5) Bus Buffer

The bus buffer consists of U1 (74LS373 - octal transparent latch), U2 (74LS244 - octal tri-state bus driver), U11 (74LS373), U12 (74LS245 - octal tri-state bus transceiver), and U13 (74LS244). The 74LS373 at U1 is used as a latch for the high order address/status lines A16/S3, A17/S4, A18/S5, A19/S6, DT/$\overline{R}$, $\overline{SSØ}$, and $\overline{IO}$/M. The 74LS244 at U2 serves as a bus driver for the eight address lines A8 through A15. The 74LS373 at U11 acts as a latch for the eight multiplexed address/data lines ADØ through AD7. Because the system uses multiplexed bus configuration, pin 25 of 8Ø88 (ALE - Address Latch Enable) is connected to the clock inputs (pin 11) of the two latches at board location U1 and U11 through pin 13 of the bus driver at U13. With the ALE line connecting to the two address latches at U1 and U11, valid address can be latched at the first T state of a bus cycle as soon as the ALE signal is pulled high. Pin 3, 8, and 18 of U1 are connected to $\overline{IO}$/M, DT/$\overline{R}$, and SSØ; since they are combined to reflect the state of system bus cycles. They also determine the state of the red LED, which is illuminated when the system is in a HALT state.

6) Memory and I/O Device Decoders

a. The ROM/RAM Decoder

The 74LS139 (dual 2 to 4 line decoder) at U14, 74LS138 (3 to 8 line decoder) at U7, and the 74LS138s at U16 and U15 are used as memory and I/O device decoders. The line decoder at <u>U14</u> is the RAM/ROM decoder. Pins 13 and 14 of the line decoder, together with pin 6 of U6, determine whether ROM or RAM is to be selected.

b. The ROM Decoder

The ROM decoder is located at U7. Pins 3 and 6 (address lines A16 and A17), pins 1 and 2 (A14 and A15), and pin 5 of the ROM decoder determine the states of the three outputs Y5, Y6, and Y7 (pins 7, 9, and 1Ø) of U7, which in turn govern which ROM chip is selected. Either two 8K x 8 or 16K x 8 ROM chips can be used as system ROM. The standard MPF-I/88 is built with one 16K x 8 ROM chips. Thus, the standard MPF-I/88 has a total memory capacity of 16K.

If A16 = 1, A15 = Ø and A14 = 1, then Y5 is pulled low. When Y5 is pulled low, the ROM chip installed at U2Ø is enabled. The starting address of this ROM chip is F4ØØØ.

If A16 = 1, A15 = 1 and A14 = Ø, then Y6 is pulled low. When Y6 is pulled low, the ROM chip installed at U19 is enabled. The starting address of this ROM chip is F8ØØØ.

If A16 = 1, A15 = 1 and A14 = 1, then Y7 is pulled low. When Y7 is pulled low, the ROM chip installed at U18 is enabled. The starting address of this ROM chip is FC000.

c. The RAM Decoder

The RAM decoder is located at U16. The state of pin 4 of U16 is determined by A16 and A17. The state of pin 5 of U16 is determined by A13 and A14. The outputs of U16, Y0, Y1, and Y2 - are determined by pins 4, 5, 2 (A12), 1 (A11), and 3 (A15). Either 2K x 8 or 8K x 8 RAM can be used as system RAM. If 2K RAM is used, the RAM decoding is shown as follows:

If zero is present on A15, A14, A13, A12, and A11, then Y0 is selected. In this case, the starting address of the RAM selected is 00000.

If zero is present on A15, A14, A13, and A12 but with A11 = 1, then Y1 is selected. In this case, the starting address of the RAM selected is 00800.

If A15 = 0, A14 = 0, A13 = 0, A12 = 1 and A11 = 0, then Y2 is selected. In this case, the starting address of the RAM selected is 01000.

If 8K RAM is used, the RAM decoding is shown as follows:

If A15 = 0, A14 = 0, and A13 = 0, then Y0 is selected. In this case, the starting address of the RAM selected is 00000.

If A15 = 0, A14 = 0, A13 = 1, then Y1 is selected. In this case, the starting address of the RAM selected is 02000.

If A15 = 0, A14 = 1, and A13 = 0, then Y2 is selected. In this case, the starting address of the RAM selected is 04000.

If 8K RAM is used, JP3 and JP4 should be re-routed as follows:

	JP4	JP3
2K x 8	Closed	Open
8K x 8	Open	Closed

d. The I/O Decoder

The I/O decoder is located at U15. Pins 1, 2, 3 (A5 through A7), 4, 6 (A8, A9), and 5 ($\overline{\text{IOR}}$ or $\overline{\text{IOW}}$) of the I/O decoder are used to determine the I/O decoding. Devices accessed through U15 are: 1) the display (I/O port address 1A0), 2) the keyboard (I/O addresses 160, 180, and 1C0), 3) the printer (I/O address 1EØ), and 4) the audio interface (I/O addresses 18Ø and 1CØ). When an I/O port is selected, the corresponding output of the decoder (Y3, Y4, Y5, Y6, Y7) is activated.

SYSTEM MEMORY

The system ROM chips are located at U18, U19, and U2Ø, while the RAM chips are located at U23, U24, and U25. Either 8K or 16K ROMs can be installed at locations U18, U19, and U2Ø. Either 2K or 8K RAMs can be installed at locations U23, U24, and U25. Jumper wires should be applied to U7 or U16 in order to select the type of RAM chip used.

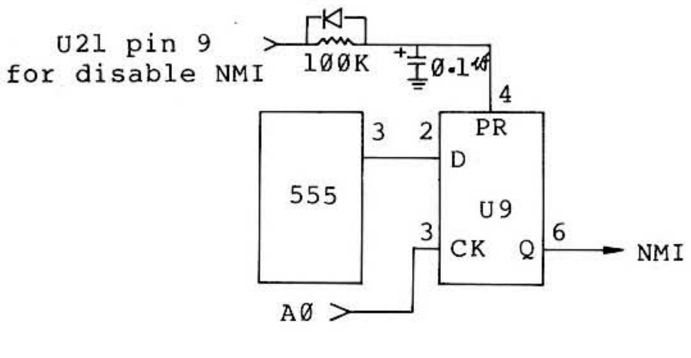
INPUT/OUTPUT INTERFACE LOGIC

The outputs of U15 (Y3 through Y7) determine exactly which I/O port is accessed. U21 and U22 are used as the latches for keyboard output data, while U26 is used as the driver for keyboard and tape input data. U27 and U8 are used for processing input signals from the Tape. Tape output signals are sent out from the output of U21 (pin 5). This pin also determines the state of the buzzer and the green LED.

Pin 3 of U28 (555) generates a 15 ms clock as the source signal for NMI. Pin 4 of U9 is programmable. It can be strapped low to disable an NMI request from pin 4 of U9 when it is desired that an NMI from this pin not to be generated.

U17 is used as a latch for printer output data.

The 78Ø5 is a voltage regulator that converts +9V input to +5V output. The +5V voltage needs to be supplied to the system for proper operation. A switching power supply must be used to supply the needed power when expansion card is to be installed to the system.



Chapter 5

Description of I/O Device Drivers

5.1 Cassette Output Device Driver

Without a device driver for writing data to tape, you have no way to store data onto tape even if the hardware circuit supports an audio output interface.

Before discussing the tape write device driver, we will describe the relationship between the tape write device driver and the command interpreter, which will affect the way the device driver executes.

COMMAND INTERPRETER

The MPF-I/88 monitor program contains a command interpreter, which prepares a user-entered monitor command in such a way that it becomes easier for the monitor to process the entered command.

A monitor command is always entered with the command character. For example, if a tape write is to be performed, then the command character is W. Sometimes a command is entered with addresses and user-specified information. For example, if you intend to write information to tape, the command line may appear as follows:

```
>W 100:00 80 /'TEST
```

Once this command line is entered, the command interpreter will count the number of addresses contained in the command line and store this number into CH. It will also count the number of bytes entered as user-specified information and load the number into CL. In the above example, the number of bytes is four since each of the character in a filename takes one byte.

Each time a monitor command is entered, the command interpreter will be called. When being called, the command interpreter will process the command line entered in the manner described above and then pass the necessary information of the command interpretation process and control to the individual monitor command for further processing.

SOME BASIC MACRO ASSEMBLER DIRECTIVES

Before going any further to explain the tape write device driver, it is necessary to pause for a while to study the Macro Assembler directives since the monitor program was assembled using Microsoft's Macro Assembler. In order to trace the monitor program thoroughly, you must be familiar with the use of the Macro Assembler.

The PROC Assembler Directive

The tape output driver W_CMD begins with the MS-DOS Macro Assembler directive PROC (short for procedure). The PROC directive is used to make the program more readable to users. During program assembly time, it tells the assembler that a whole PROC block is to follow. In other words, a block of assembly program instructions will follow the PROC assembler directive. A PROC is executed from a CALL or JMP instruction. For more details of the MS-DOS assembler directives, you can refer to Microsoft's Macro Assembler Manual. If you don't have that manual, consult your MPF-I/88 distributor for information on how to purchase that manual.

The W_CMD procedure contains the following important procedures:

FILE_WRITE	: Write MPF-I/88 tape format to tape.
TAPE_WRITE	: Write IBM PC tape format to tape.
WRITE_1_BYTE	: Write one byte to tape.
WRITE_1_BIT	: Write one bit to tape.

The functions of these procedures will be explained later. After reading the descriptions of these procedures, you are suggested to trace these procedures instruction by instruction. Tracing a program is the best way to learn programming.

Now you are suggested to find the W_CMD procedure in the MPF-I/88 Monitor Program Source Listing. To get to know how to read the monitor source program, you need to refer to the Microsoft's MS-DOS Operating System Macro Assembler Manual and Microsoft's Cross-Reference Utility for MS-DOS Operating System. If you do not know how to get these two manuals, please consult your MPF-I/88 distributor. But even if you do not have the two manuals at hand, we will still teach you how to read the monitor source program.

To find the W_CMD procedure, you need to use the cross reference section of the monitor source program listing. The first page of the monitor source program listing comes under the heading

```
 The Microsoft MACRO Assembler, Version 1.25  Page 1-1
```

That message says that the monitor source program was assembled using Microsoft's MACRO Assembler, Version 1.25. Since there are several different versions for the Macro Assembler, it is important to note the version number in order to distinguish among different versions. Page number is printed together with the heading on each page for easy reference. What comes on the next line following the heading is the date it tells when the monitor source program was assembled. A general practice is that a monitor program will have to be assembled for many times before it is finally released. From the program listing of the MPF-I/88 monitor program, you will know that the current release of the monitor program is based on the source program which was assembled on Jan. 17, 1985. Sometimes it is possible for a company to upgrade the software without prior notice.

SYMBOL TABLE

Thumbing through the monitor source program listing, you will discover that there are 78 pages which are printed under the same heading. Then you will come across the part designated as the symbol table for the source program you have just gone though. The symbol table lists all the symbols used in the program and gives such information as type, value, and attribute related to a symbol. Please refer to Microsoft's Macro Assembler Manual for details. The symbol table comes under the heading:

The Microsoft MACRO Assembler, Version 1.25 Page Symbols-1

You will find that there are a total of 14 pages of symbol table.

CROSS REFERENCE

Then comes the cross reference section which is printed under the heading:

Symbol Cross Reference (# is definition) Cref-1

You will find that there are a total of 14 pages of cross reference.

The most efficient way to find a routine in the source program such as W_CMD is to use the cross reference. The entries in the cross reference section are listed alphabetically. To find the location of the procedure W_CMD, you should go through the entries until you found W_CMD. On page 14 (Cref-14) you can locate the entry of W_CMD. It is listed as follows:

```
W_CMD ................ 2940#  2991  4083
```

The three numbers following the procedure name W_CMD are the line numbers affixed to each program line in the monitor source program listing by the Macro Assembler. Note that each line of the monitor source program listing is prefixed with a line number. The three numbers are where you can find the name W_CMD. The line number with a # sign is where the name W_CMD is defined. To find out how W_CMD works, you should refer to line 2940 which is located on page 1-54.

The ASSUME Assembler Directive

Following the CLI instruction is the assembler directive ASSUME. This directive tells the Macro Assembler where (in which segment) symbols can be referenced. In the tape output driver program, symbols can be referenced through CS and DS registers. The code segment is pointed to by CS register and the data segment is pointed to by the DS register.

LABEL

To output a bit from the system, you must first load the DX register with the I/O port address (180H), which is specified by the label TAPE_IO_OUT. A label is a name which is converted to an address when the program is assembled by the assembler. A label is usually the destination for a JMP, CALL, or LOOP instruction.

For more detailed definition for LABEL and the use of the LABEL directive, please refer to Microsoft's Macro Assembler Manual.

The W_CMD procedure contains the following labels:

```
W_CMD_1
W_CMD_2
FILE_LEADER
WRITE_BLOCK
WRITE_CRC_BYTE
WRITE_TAILER
```

When a program is too complex to trace, you are suggested to trace the labels first and then you will be able to know the program logic, based on your understanding of labels and procedures.

Now we are going to introduce to you some basics on the write-to-tape device driver.

Bit 6 of the output port TAPE_IO_OUT is the bit from which data is written out

When information is to be output from the system, bit 6 of the port specified by TAPE_IO_OUT is used to send out the bit string.

Disable Interrupt

The DISABLE_INT routine clears the interrupt flag and NMI interrupt so that a tape write operation will not be interrupted by another event.

OUTPUT A BIT 1

When information is written to tape, actually a bit string consisting of zeroes and ones are output serially from bit 6 of TAPE_IO_OUT port.

When a one is to be output, bit 6 of port 180H actually outputs a one ms (millisecond) pulse with a high 500 ns (nanosecond) half cycle and a low 500 ns half cycle.

OUTPUT A BIT Ø

When a zero is to be output, bit 6 of port 18ØH actually outputs a Ø.5 ms (millisecond) pulse with a high 25Ø ns (nanosecond) half cycle and a low 25Ø ns half cycle.

FUNCTIONAL DESCRIPTION OF TAPE OUTPUT DRIVER

The following is a functional description of the tape output driver W_CMD.

After the command information as processed by the command interpreter is submitted to the individual command, the individual command will examine if the command is entered according to the command syntax. If it is entered according to the command syntax, a CALL or JMP instruction will be executed to perform the desired functions. If not, the command will set the Carry flag and a RET instruction will return program control to the command interpreter, which will then display the error code telling the user that the command entered is not executable because of command syntax error. Note that when an error is detected by the individual command, it will always set the Carry flag to let the command interpreter know that an error has occurred..

For the W_CMD routine, it will first check if the entered command follows the defined syntax of the command. If not, an error message will be shown. The W_CMD routine assumes that a memory range will be output to tape, thus the starting address of the memory range should always be smaller than the ending address. If the starting and ending addresses are entered otherwise, then a range incorrect error will be displayed.

FILE_NAME_FILLER -- Filler Bytes

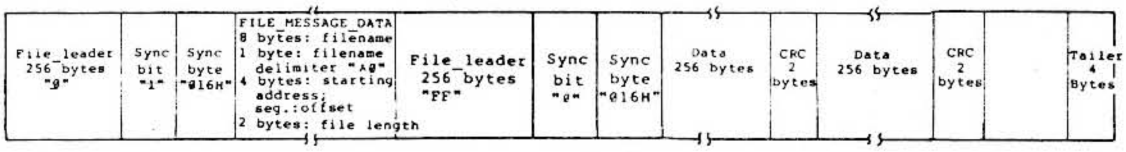
After W_CMD has performed the command syntax and the memory range checks, it will check whether the length of filename is less than eight characters. The length of a filename should never be greater than eight bytes (characters). If it is greater than eight characters, then error message will be displayed by the command interpreter. If the filename length is less than eight characters, the W_CMD routine will continue by calling the FILE_NAME_FILLER.

An 8-byte memory space is reserved for the characters which make up the filename. If less than eight characters are used, FILE_NAME_FILLER will fill the unused memory space with the ASCII code for the space character (2ØH) and execute a RET to the main program to execute W_CMD_2. W_CMD_2 will place the end of filename code (ØAØH) to the position immediately following the memory space containing the filename. The remaining instructions of W_CMD_2 are designed to prepare a set of pointers and counter such as the ES, SI, and CX. The ES and SI are loaded with the segment and offset addresses of the starting address, respectively, while CX is loaded with the value of file length.

FILE_WRITE -- Writing MPF-I/88 Tape Format to Tape

After loading the pointers and counter with appropriate values, the tape output driver will write the MPF-I/88 tape format to tape. MPF-I/88 tape format is described below:

MPF I/88 TAPE FORMAT



The MPF-I/88 tape format starts with a file leader. The file leader is 256 consecutive bytes of zeroes. The file leader is designed to let the system know that a file is about to start when data is to be read back to the system. After writing the leader to tape, the tape output driver will write a sync bit 1 and a sync byte 16H, which is followed by the filename, starting address of the memory range to be output, and file length, to tape.

Writing a Ø.2 Second Delay to Tape

Since the tape input device driver is designed to be able to read information stored in IBM Personal Computer tape format, the MPF-I/88 tape output driver will also write the IBM PC tape format to tape with the TAPE_WRITE procedure. But before writing the IBM PC tape format to tape, a Ø.2 second delay is output to tape to separate the MPF-I/88 and IBM PC tape format.

TAPE_WRITE -- Writing Data Block to Tape

After writing the Ø.2 second delay, the tape output device driver will write data block to tape.

WRITE_BLOCK

This block of instructions (sometimes a block of instructions is also called a program module) performs the actual data output operation. It calls WRITE_BYTE, and WRITE_1_BYTE in turn calls WRITE_1_BIT in order to output data to tape.

WRITE_FILLER_BYTE

Data is written to tapė in units of 256 bytes. In other words, 256 bytes form a data record. If the data to be recorded unto tape is less than 256 bytes, the unused bytes are filled with filler bytes, which is meaningless to the system when they are read back from tape. Since one data record is insufficient for recording the tape format, the unused area of the second data record is filled with filler bytes.

WRITE_1_BIT and WRITE_1_BYTE

Data is written to tape one bit at a time. The data bit to be output is first placed in the Carry flag and then output to bit 6 of port TAPE_IO_OUT. One byte of data is output by using the LOOP WRITE_ALL_BIT instruction.

WRITE_CRC_BYTE

When WRITE_1_BYTE is executed, the subroutine CRC_GEN (CRC byte generator) is called. CRC_GEN is called to generate the values to be placed in the two CRC bytes. After 256 bytes have been output to tape, WRITE_CRC_BYTE will write two CRC bytes to tape.

WRITE_TAILER

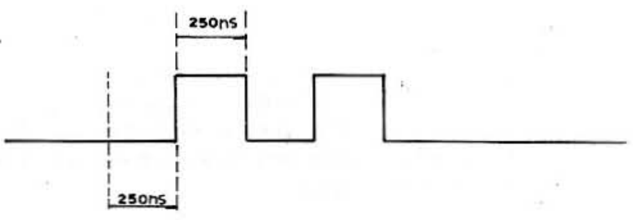
After the whole memory range is output to tape, a file tailer will be output to tape by WRITE_TAILER. The file tailer consists of four bytes of 1.

A CLOSER LOOK OF WRITE_1_BIT

Although we assume that at this time you have cultivated the habit of tracing the instructions of a program in
order to follow the logic flow of a program, we still feel you may be interested in some of the programming techniques applied to write the tape output driver. We will trace the WRITE_1_BIT procedure in more detail below.

DISPLAY_25Ø

After PUSHing CX and AX onto the system stack (This is for saving the values of CX and AX) for future use, since the values of these two registers will be altered in the WRITE_1_BIT procedure), the value of the variable DISPLAY_25Ø (39 = 27H) is loaded into CX. This value and TUNING_1 (17 = 11H) make sure that when a zero is output, the pulse wave for a zero will consist of a high 25Ø ns half cycle and a low 25Ø ns half cycle as illustrated below:



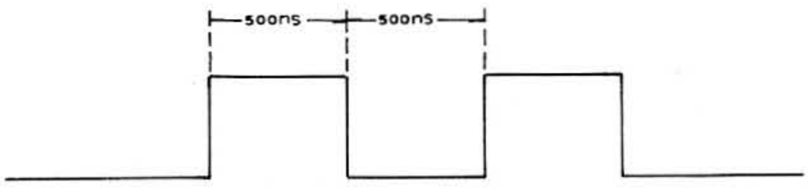
PULSE WAVE FOR A BIT Ø

Note that ØCØH is loaded into AL in the first instruction of W_BIT_Ø. This value represents a bit pattern of 11ØØØØØØ. This bit pattern is then output to port TAPE_IO_OUT which is addressed by DX. Note bits 7 and 6 are both one at this time. Bit 6 is used to access the TAPE_IO_OUT port. Bit 7 actually has nothing to do with tape output driver. However, if bit 7 is set to Ø, then you won't be able to activate the printer when you intend to access the printer later. This is because that bit 7 of port 18ØH is used for printer strobe.

AL is ANDed with the value ØBFH in order to set bit 6 of TAPE_IO_OUT to zero. After bit 6 is set to zero as a result of the AND operation, the bit pattern 1Ø111111 is output to TAPE_IO_OUT using the OUT instruction. This begins the low 25Ø ns half cycle of a zero pulse wave.

The Carry Flag

The instruction JNC W_BIT_Ø A in the WRITE_1_BIT procedure is used to determine if a bit Ø is to be output to tape. If it is, program execution will flow to W_BIT_Ø as we have just mentioned. If the carry flag is set, then a bit 1 is to be output to tape and W_BIT_1 will be executed. Note that when a bit 1 is to be output to tape, the time delay for the LOOP operation will be lengthened by adding DISPLAY_25Ø to TUNING_2 (61 = 3DH). This is because a bit 1 takes a high 5ØØ ns half cycle and a low 5ØØ ns half cycle to represent. The pulse wave for a bit 1 is illustrated as follows:



The values for DISPLAY_25Ø, TUNING_1, and TUNING_2 are caculated by summing up the execution time of each instruction involved in a WRITE_1_BIT operation. You can try to figure out how to calculate these values as an exercise.

5.2 Cassette Input Device Driver

Without a device driver for reading data from tape, you have no way to access data which is stored on tape even if the information was previously stored on tape with a tape output (write-to-tape) device driver such as the one we have mentioned in the previous chapter.

If you have already traced the instructions in the previous experiment, then the read-from-tape device driver to be discussed will be easy for you to understand.

Instead of discussing the instructions one by one, we will study the device driver modularly. In other words, the monitor command R (or the R_CMD procedure) is discussed according to the functions of each procedure used in the tape input device driver.

The device driver allows you to read MPF-I/88 or IBM PC formatted tape. However, if you intend to load a tape of IBM PC tape format to the memory of MPF-I/88, you must make sure there is enough amount of RAM for the program to be loaded.

You are suggested to read the chapter on I/O Programming of this manual in order to get some basic I/O programming concepts before reading the following paragraphs any further. You are also suggested to trace the instructions of the procedures carefully as listed in MPF-I/88 Monitor Program Source Listing in order to learn the art of 8Ø88 assembly language programming. Tracing a program can be one of the best ways to learn programming.

After reading the chapter on I/O Programming and open up your MPF-I/88 Monitor Program Source Listing, you are ready to read further.

The device driver (procedure R_CMD) contains the following procedures:

```
FILE_READ
TAPE_READ
READ_BLOCK
READ_1_BYTE
READ_1_BIT
READ_HALF_BIT
```

A smart way to learn programming is to trace a program modularly. You are suggested to try to figure out the function of each procedure and then the function of labels contained in the R_CMD procedure.

If a procedure is too complex to trace, examine the functions of labels related to the procedure first and then you will have some ideas of how the procedure works to complete a specific task. This is the kind of decipline that good programmers need.

LABEL

A label is a name that serves as the target of LOOP, JUMP, and CALL instructions. In other words, a label is used as the operand for LOOP, JUMP, and CALL instructions. A label is assigned an address by the assembler. A label is entered by the program in the source program. After the source program has been assembled, labels are converted to addresses by the assembler. Please refer to Microsoft's Macro Assembler Manual for more details about label.

FUNCTIONAL DESCRIPTION OF THE TAPE-READ DEVICE DRIVER

The following is a functional description of the tape-read device driver.

Check If a Command Line Is Entered Correctly

To read data from tape, the tape input device driver first checks if the command line was entered without syntax error and whether a legal filename was entered.

As you may recall, the command interpreter will submit some data to the R command (the read-from-tape device driver). The case is similar to the W command. In case a command line is entered as follows:

```
>R <addr>/<filename>
```

The command interpreter will store the number of addresses entered in CH and the number of characters which make up the filename in CL.

Two CMP instructions are used to check if the command line was entered without syntax error and whether a legal filename was entered. If an error is detected, the command interpreter will display the corresponding error code of that error.

If the command line is entered correctly, the device driver will execute the FILE_READ procedure to fetch the MPF-I/88 file leader, including the sync bit, sync byte, etc.

Since data is written to tape in a pre-defined tape format as mentioned in the previous experiment and Chapter 8, I/O Programming, of the MPF-I/88 User's Manual, data is read back into the system according to the same tape format. Thus, after MPF-I/88 file leader has been read from tape, the device driver will execute procedure TAPE_READ to fetch the IBM PC tape leader.

After the IBM PC file leader has been fetched, the device driver will execute the procedure READ_BLOCK to fetch the 256-byte data record and the accompanying CRC bytes.

After all the data records and the accompanying CRC bytes have been read back to system memory, the device driver will execute procedure READ_TAILER to fetch the four tailer bytes to complete the R_CMD procedure.

Unlike the W_CMD which writes to tape one bit at a time using procedure WRITE_1_BIT, the most critical procedure contained in the R_CMD procedure is READ_HALF_BIT.

A CLOSER LOOK OF READ_HALF_BIT

The instruction IN AL,DX is used to read data from bit 7 of input port TAPE_IO_IN (1CØH) to system. As you may remember, a bit Ø is the equivalent of a pulse whose pulse width is 5ØØ ns (consisting of a low 25Ø ns half cycle and a high 25Ø ns half cycle) while a bit 1 is a pulse with a pulse width of Ø.5 ms (consisting of a low 5ØØ ns half cycle and a high 5ØØ ns half cycle). A low is sensed from bit 7 of the tape input port 1CØH (using IN AL,DX) is when nothing is sent from tape. Once a high is sensed, it means either a bit Ø or a bit 1 is read from tape.

Detecting a High from Bit 7 of the Tape Input Port

The instruction XOR AL,TAPE_STATUS does the job.

TAPE_STATUS is a memory location which is assigned with the variable name TAPE_STATUS by the DB (Define Byte) assembler directive.

The DB assembler directive tells the assember to reserve a memory space (which is identified by the variable name TAPE_STATUS) for a value, which may be altered during program execution.

TAPE_STATUS, as its name implies, is used to signal the tape status. If a high is sensed from bit 7 of the tape input port, the contents of this variable are set to 1. If a low is sensed, the value of this variable is set to Ø.

Upon system initialization, the value of TAPE_STATUS is cleared to Ø. If AL contains a zero, then the zero flag is set and the instruction JS READ_NEXT_STATUS will cause READ_NEXT_STATUS to be executed again in order to detect a low-to-high transition of bit 7 of tape input port. If a non-zero value is stored in AL, then it means that a low-to-high transition occurs at bit 7 of the tape input port. After this low-to-high transition is detected, the value of TAPE_STATUS is altered.

When a low-to-high transition is detected at bit 7 of the tape input port, it means that either a zero or a one has been read by the system.

But how does the system distinguish between a bit Ø and a bit 1?

The instruction OR CX,CX does this job. CX contains the value specified by 2 x DELAY_375. This value is ORed with itself in order to detect if a zero is contained in CX. If CX contains a zero, it means the counter CX has counted to zero when TAPE_STATUS is changed. If this is the case, a one was read from tape to system. If the Sign flag is not set, it means a non-zero result is in CX (this indicates that a low-to-high transition occurred before the value in CX was decremented to zero), In this case, a bit Ø is read from tape to system.

It is the counter value stored in CX that determines if a bit Ø or bit 1 was read from tape. This value is derived from summing up the execution time of the related instructions.

By storing an appropriate value in CX, you can detect whether a bit Ø or a bit 1 is read from tape in a half cycle.

5.3 RS-232-C Interface Driver

When transmitting data, it can be transmitted serially (one bit at a time) or in parallel (eight bit a time). Data is usually transferred to a near-by printer in parallel. But data is transmitted to a remote work station or a computer network via a serial communications link such as a telephone line.

When two devices are installed next to each other, then it is much faster to transmit data in parallel than serially. However, serial data transmission is often used for data communications. This is because when data is to be transmitted to a remote place, using serial communications line is much more economical than using parallel data communications lines.

The major drawback of serial communications is that it takes a longer period of time to transmit the same amount of data as compared with parallel communications.

THE EIA RS232-C INTERFACE

Most popular microcomputers support serial communications with built-in or optional serial communications ports. Currently there are several common serial communications interfaces being used. The most popular serial communications interface is RS232-C as set forth in the Electronics Industries Association standard.

CONTROL SIGNALS

Start Bit

In a serial communications link, data is sent out one bit at a time together with control information. When the system is sending out data, it must have a way to tell the receiving device that when the data will be transmitted. In reality, the system will transmit a start bit when data is to be transmitted. A start bit is usually a logical Ø on the transmission line. In this case, the transmission line is said to be in the spacing state.

Stop Bit

When a data transmission has been completed, the system must tell the receiving device that the transmission has completed. This is done by sending stop bit(s) to the receiver. There can be 1, 1.5, or 2 bits depending on the exact data transmission environment. After stop bit has been received, the receiving device does not look forward to receive data from the transmission line unless another start bit is received. A stop bit is normally a logical high on the transmission line. When the transmission line is logical high, it is said to be in a marking state.

Parity Bit

When the data communications line is very long, you can add a parity bit for each character to be transmitted. Parity bit is added to ensure the accuracy in data transmission. The parity bit may be a Ø or a 1. If even parity check is selected, then the number of 1 bits which make up the data bits and parity bit must be even. If odd parity check is selected, then the number of 1 bits which make up the data bits and parity bit must be odd.

Data Bits

The data bits are transmitted to the receiving device following the start bit. There can be 5, 6, 7 or 8 data bits. The number of data bits must be consistent in the same data transmission. But the number of data bits may not be fixed in each data transmission. Data bits are transmitted least significant bit first. By not fixing the number of data bits, the transmission can be speeded up.

The Baud Rate

The data transmission speed is measured in bits per second (bps). It is referred to as the baud rate. If a device is said to operate at 96ØØ baud, it actually transmit or receive bit string at 96ØØ bits per second.

THE 825Ø ASYNCHRONOUS COMMUNICATIONS ELEMENT

The job of converting data into a bit string together with control information would be quite time consuming and difficult for human beings. Thus, a special-purpose microprocessor is designed to handle serial data communications - the 825Ø.

The 825Ø can be programmed easily to handle serial data communications. The 825Ø must be initialized before being used. That is to say you have to tell the 825Ø (by using the OUT instruction) the desired baud rate, the number of data bits and stop bits, and the type of parity check. such information is generally known as serial communications protocol.

Please refer to the data sheet provided by the manufacturer of the 825Ø async communications element for more details.

The following is a description of a routine for doing RS232-C serial communications. It is a subroutine contained in MPF-I/88 monitor program. You can use that routine in your own program in order to perform RS232-C serial communications. Or, you can design your own RS232-C serial communications routine after you have become familiar with RS232-C serial communications programming. You can use the instruction INT 13H to use that routine. But before invoking that routine by entering the INT 13H instruction, you should load appropriate values (usually referred to as input parameters) into the proper 8Ø88 registers. The input parameters are then passed to the appropriate registers in the 825Ø.

The RS232-C routine, also called RS232-C device driver, performs the following four functions:

1) Initializes the 825Ø.
2) Transmits data - one character at a time.
3) Receives data - one character at a time.
4) Read the status of the 825Ø.

The RS232-C device driver can be divided into four modules or blocks. Each module performs a specific function as described above. The initialization function is identified by the lable FUNØ in the program listing. The character transmission function is identified by the label FUN_1, while the character receive function by FUN_2. The status read function is identified by the label FUN_3.

The device driver starts with saving the current state of DX, BX, and DS registers by pushing their contents onto the stack. The fourth instruction CALL CDS sets the contents of data segment to zero. By setting the value of DS to zero, the data stored in the first 2K system memory for system use (Ø:Ø to Ø:7FF) can then be accessed by the RS232-C device driver. The sixth instruction loads zero into the counter TIME_COUNT. Since the counter is located in memory location Ø:51Ø, the device driver won't be able to access the counter unless DS points to zero.

Since the AX register will be used for passing input parameters to the asynchronous communications element 825Ø, the contents are loaded into the BX register for temporary storage in the fifth instruction, which is located in the offset address FCEØH in the code segment.

The seventh and eighth instructions - CMP AH,3 and JA R2Ø - are designed to determine if a legal function call is made. If the value stored in AH is greater than 3, than a jump instruction is executed to return the control to the calling program.

If the zero flag is set, it means that the value of AH is 3. When AH = 3, the module (function) for returning 825Ø status will be executed.

The 1Øth and 11th instructions test if Ø is stored in AH. If it is, the sign status is set to 1 and a jump instruction will cause the function FUNØ to be executed.

The 12th and 13th instructions test if 1 is stored in AH. If it is, the sign status is set to 1 and a jump instruction will cause the function FUN1 to be executed.

If the above jump instructions are not executed, then it is obvious 2 is stored in AH. If this is the case, FUN_2 is executed. As you may still remember, FUN_2 is responsible for receiving data from a RS232-C device. Let's examine how this is done by the RS232-C device driver.

INPUT A CHARACTER FROM AN RS232-C DEVICE

When data is to be input from an RS232-C device, a message should be output to the transmission device telling the transmission device that the system is ready to receive data. The message should be sent to the modem control register of the transmitting 825Ø.

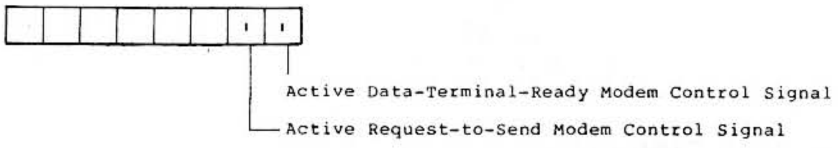
To send information to a register inside 825Ø, you must know the address of that register. Two sets of I/O port addresses can be assigned to the registers inside 825Ø. The first set of I/O port addresses that can be assigned to 825Ø registers ranges from 3F8H through 3FEH, while the second set of I/O port addresses which can be assigned to 825Ø regiters starts from 2F8H through 2FEH. The I/O port addresses assigned to 825Ø registers are listed as follows:

I/O Port Address	Input or Output	Register
3F8H	Output	Transmitter holding register
3F8H	Input	Receiver data register
3F8H	Output	Baud rate divisor (LSB)
3F9H	Output	Baud rate divisor (MSB)
3F9H	Output	Interrupt-enable register
3FAH	Input	Interrupt-identification register
3FBH	Output	Line-control register
3FCH	Output	Modem-control register
3FDH	Input	Line-status register
3FEH	Input	Modem-status register

As you can see from the above table, the I/O port address for the modem control register is 3FCH. Since the DX is loaded with the lowest port address assigned to 825Ø registers, the first instruction in the FUN_2 module adds 4 to DX (which contains 3F8H) in order to access the modem control register.

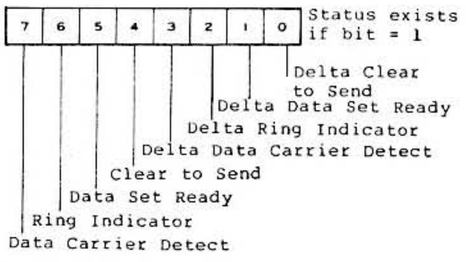
Actually two signals are sent to the modem control register -- data terminal ready (DTR) and request to send (RTS). The two signals are sent to the modem control register by outputing the value 3 through AL register.

After sending the two signals to the transmitting device, bit Ø and bit 1 of the modem control register are set to 1. This is illustrated as follows:


1
1
Active Data-Terminal-Ready Modem Control Signal
Active Request-to-Send Modem Control Signal

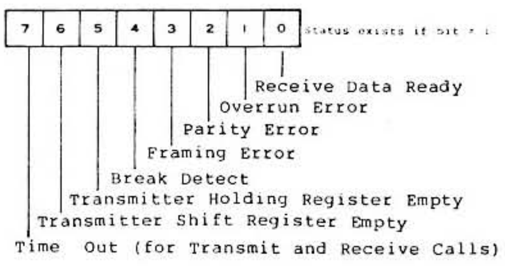
*** Modem Control Register ***

Before receiving information from the transmitting device, you must also make sure that the transmitting device is ready to send information. This can be done by reading the modem status register, which is assigned port address 3FEH. The modem status register contains eight bits with each bit signaling a specific status. The modem status register is illustrated as follows:



*** Modem Status Register ***

To make sure if the transmitting device is ready, we check whether bits 4 and 5 are set to 1. If they are set, i.e., data set ready and clear to send, the device driver will check the next condition - if bit Ø of the line status register is set. If it is set, then a character can be input from the transmitting device. If bit Ø of the line status register is not set, the device driver will keep testing bit Ø of the line status register until it is set to 1. The line status register is shown as follows:


7 6 5 4 3 2 1 0
Status exists if bit = 1
Receive Data Ready
Overrun Error
Parity Error
Framing Error
Break Detect
Transmitter Holding Register Empty
Transmitter Shift Register Empty
Time Out (for Transmit and Receive Calls)

*** Line Status Register ***

If bits 4 and 5 are not set, the RS232-C device driver will call the CHK_TIME subroutine. The counter TIME_COUNT is decremented by CHK_TIME subroutine. If the counter is not decremented to zero, the device driver will loop back to check bits 4 and 5 of modem status register. If bits 4 and 5 are set, the device driver will check bit Ø of line status register, if that bit is set, then a character will be transmitted from the transmitting device to the system.

If the counter TIME_COUNT is decremented to zero, it is assumed that no data will be sent to the receiving device and a jump instruction will cause CHK_TIME_1 to be executed. This subroutine will set the Carry flag and then execute a RET instruction. After the RET instruction has been executed, the TIMEOUT subroutine will be excuted. The TIMEOUT subroutine will make another jump to IN_STATUS before returning the control to the calling program.

READ THE STATUS OF 825Ø

FUN_3 is used to examine the status of 825Ø. After making this function call, AH will contain the contents of line status register, and AL will contain the contents of modem status register.

The first few instructions load zero into CH, and then add 4 to DX so that DX will point to the line status register. Note that the instruction MOV CH,Ø is used to clear the contents of CH to Ø. This instruction, together with OR AL,CH and MOV AH,AL, sets bit 7 of the line status register to zero. When bit 7 of the line status register is zero, time-out won't occur. The contents of line status register are first input to AL. After the line statuses are ORed with the contents of CH (zero), the results are moved to AH. At this time, AH contains the line statuses.

The contents of CH is then ORed with themselves. This instruction is here in order to set the zero flag for future use by the JNZ RTS instruction. If the zero flag is set, the original contents of AL, which was moved to BL in the fifth instruction of the RS232-C device driver, are loaded from BL to AL. Then program control will be returned to the calling program. If the zero flag is not set by the OR instruction, DX will be incremented to point to the modem status register. The IN AL,DX instruction is then used to return modem statuses to AL.

INITIALIZE THE 825Ø

To initialize 825Ø, you have to load AH with zero, AL with the desired parameters, and DX with port address 3F8H.

An AND instruction is placed in the beginning of FUN_Ø to isolate the three most significant bits. In other words, this instruction ignores the state of bit Ø through bit 4 contained in AL. Then CL, which is used as a counter here, is loaded with five. The contents of AL are then shifted right five times. After the shift operation, the contents of AL are loaded into CL.

We will pause here for a while to study how serial communications protocol is loaded into 825Ø. The initialization will affect the following registers in 825Ø:

1) Baud rate divisor (LSB) - Port address 3F8H
2) Baud rate divisor (MSB) - Port address 3F9H
3) Line control register - Port address 3FBH
4) Interrupt enable register - Port address 3F9H

Initializing the Baud Rate Divisor Registers

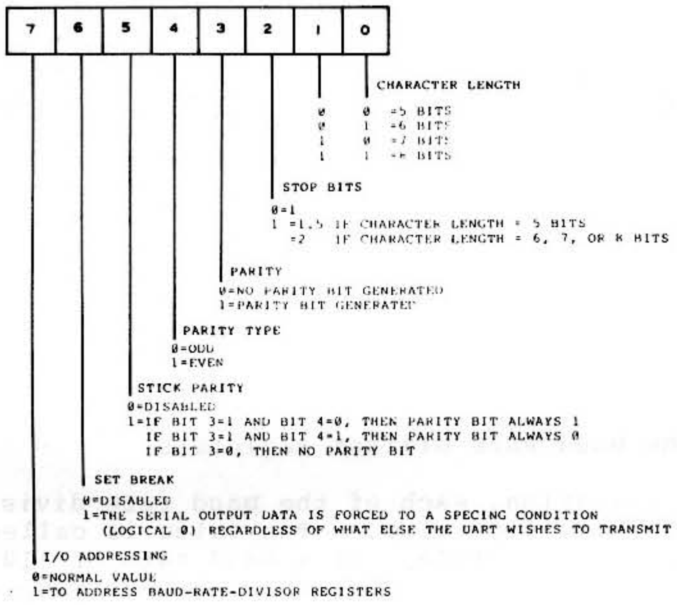
After the initialization, each of the baud rate divisor registers is loaded with a specific value. The value is called baud rate divisor value. For example, if a baud rate of 11Ø is desired, Ø4H is loaded into baud rate divisor register (MSB) and 17H is loaded into baud rate divisor register (LSB). If a baud rate of 15Ø is desired, Ø3H is loaded into baud rate divisor register (MSB) and ØØH is loaded into baud rate divisor register (LSB). The relationship of the desired baud rates and their corresponding baud rate divisor values are listed as follows:

*** Table of Baud Rate Divisor Values ***

Desired Baud Rate	Value for Baud-Rate-Divisor Registers	
	MSB	LSB
5Ø	Ø9H	ØØH
75	Ø6H	ØØH
11Ø	Ø4H	17H
134.5	Ø3H	59H
15Ø	Ø3H	ØØH
3ØØ	Ø1H	8ØH
6ØØ	ØØH	CØH
12ØØ	ØØH	6ØH
18ØØ	ØØH	4ØH
2ØØØ	ØØH	3AH
24ØØ	ØØH	3ØH
36ØØ	ØØH	2ØH
48ØØ	ØØH	18H
72ØØ	ØØH	1ØH
96ØØ	ØØH	ØCH

Initializing the Line Control Register

The function call FUN_Ø will also load information on the type of parity, stop bit, and character length to the line control register. The function of each bit in the line control register is briefly described in the following diagram.



7
6
5
4
3
2
1
0
CHARACTER LENGTH
Ø Ø =5 BITS
Ø 1 =6 BITS
1 Ø =7 BITS
1 1 =8 BITS
STOP BITS
Ø=1
1 =1.5 IF CHARACTER LENGTH = 5 BITS
=2 IF CHARACTER LENGTH = 6, 7, OR 8 BITS
PARITY
Ø=NO PARITY BIT GENERATED
1=PARITY BIT GENERATED
PARITY TYPE
Ø=ODD
1=EVEN
STICK PARITY
Ø=DISABLED
1=IF BIT 3=1 AND BIT 4=Ø, THEN PARITY BIT ALWAYS 1
IF BIT 3=1 AND BIT 4=1, THEN PARITY BIT ALWAYS Ø
IF BIT 3=Ø, THEN NO PARITY BIT
SET BREAK
Ø=DISABLED
1=THE SERIAL OUTPUT DATA IS FORCED TO A SPECING CONDITION
(LOGICAL Ø) REGARDLESS OF WHAT ELSE THE UART WISHES TO TRANSMIT
I/O ADDRESSING
Ø=NORMAL VALUE
1=TO ADDRESS BAUD-RATE-DIVISOR REGISTERS

*** Diagram of Line Control Register ***

The line control register is initialized in our function call with the OUT DX,AL instruction. Before this instruction is executed, the contents of AL is anded with a bit mask 1FH in order to zero out the first three most significant bits.

Initializing the Interrupt Enable Register

After the line control register is initialized, the function call will initialize (disable) the interrupt enable register. Handling serial communications with interrupt would be very complex. Since the use of interrupts is not necessary for serial communications, the interrupt enable register is usually disabled.

We will continue explaining the function call FUNØ. After shifing AL and loading the contents of AL to CL, the routine will determine if CL contains Ø using the OR CL,CL instruction. If it is, a jump to BAUD_OUT will be executed. Note that before the OR instruction, AX is loaded with the baud rate divisor value Ø417H = 1Ø47 (in decimal). The baud rate divisor value is then loaded into CX in preparation for use by two MOV instructions which will move the value to the baud rate divisor registers.

To access the baud rate divisor registers, bit 7 of the line control register should be set to 1. To achieve this goal, we use the instruction ADD DX,3 to make DX points to the line control register. Then the MOV AL,80 instruction and OUT DX,AL is used to set bit 7 of the line control register.

To load the baud rate divisor value to the baud rate divisor registers, we POP DX so that DX points to the baud rate divisor register (LSB). Now the LSB value is loaded to AL and OUT to DX. Then DX is incremented and the MOV and OUT instructions are used again to load the MSB baud rate value to the MSB baud rate divisor register.

Now the baud rate divisor registers have been set properly. The following five instructions are used to initialize the line control register so that 8250 will know the number of stop bits, the parity type, and character length. As you may remember, BL is actually stored with the original value of AL -- the input parameter. We will move this value to AL and use a bit mask 1FH to eliminate the first three most significant bits -- those bits used to specify the baud rate. The AND operation performs this job. After the AND operation, AL only contains such information as the number of stop bits, the parity type, and character length. After incrementing DX so that DX points to the line control register, an OUT instruction is used to load the line control register with appropriate serial communications protocol.

Now we are going to disable the interrupt enable register, which can be disabled by setting its value to zero. We first decrement DX so that the value of DX points to the interrupt enable register. Then we use the XOR instruction to zero out AL. By using the OUT DX,AL instruction, zero are sent to the interrupt enable register.

Now that the 8250 has been initialized, the IN_STATUS routine will be executed to return serial communications statuses to AX.

OUTPUT A CHARACTER -- FUNCTION 1

When you intend to output a character through the serial communications line, you must load AH, AL, and DX with appropriate values. This is listed as follows:

1) AH = 1
2) AL = The character to be transmitted.
3) DX = Port address.

Function 1 will return the contents of line status register in AH if a character is transmitted successfully. If the character is not transmitted successfully, then bit 7 of AH is set to 1.

FUN_1 will first output the status of the transmitting device to modem control register. If bits Ø and 1 of the modem control register are set, it means that the transmitting device is ready to send out information.

Then it will read the status of modem status register. If both bits 4 (clear to send) and 5 (data set ready) are set, it means that the receiving device is ready to receive information.

Even after you have ensured that both the transmitting and receiving devices are ready, character still will not be transmitted unless bit 5 (transmitter holding register empty) of line status register is set. If it is set, then a character will be output to the receiving device, and program control will be returned to the calling program with the contents of line status register stored in AH.

Things may not be going that smoothly sometimes. What will happen if bits 4 and 5 of the modem status register are not set? What if bit 5 of line status register is not set as expected?

If bits 4 and 5 of the modem status register are not set

A time counter (TIME_COUNT) is designed to solve this problem. As you may remember, a zero was loaded into the counter when the RS232-C device driver was first invoked. Once FUN_1 finds out that bits 4 and 5 are not set, it will call the CHK_TIME subroutine. The CHK_TIME subroutine will decrement the time counter TIME_COUNT by one from FFFFH and check if the counter has counted to zero. If the counter has not counted to zero, FUN_1 will go back and check bits 4 and 5 again. If these two bits are set, FUN_1 will check bit 5 of the line status register. Otherwise, it will call CHK_TIME again.

If bits 4 and 5 are not set when TIME_COUNT has counted to zero, FUN_1 will jump to CHK_TIME_1, set the carry flag, and then execute TIMEOUT and jumpt to IN_STATUS so as to load the contents of line status register to AH and return program control to caller.

The counter is designed for returning program control to the calling program if bits 4 and 5 of the modem status register are not set.

If bit 5 of line status register is not set

If bit 5 of line status register is not set, FUN_1 will also call CHK_TIME, decrement the time counter TIME_OUT, and check if the contents of time counter is decremented to zero. If the counter has not counted to zero, FUN_1 will go back and check bits 4 and 5 again. If these two bits are set, FUN_1 will check bit 5 of the line status register. Otherwise, it will call CHK_TIME again.

If bits 4 and 5 are not set when TIME_COUNT has counted to zero, FUN_1 will jump to CHK_TIME_1, set the carry flag, and then execute TIMEOUT and jumpt to IN_STATUS so as to load the contents of line status register to AH and return program control to calling program.

5.4 LCD Driver

The MPF-I/88 supports a 20-column by 2-line physical LCD display. Therefore, 20 by 2, or 40 characters can be displayed on the LCD at one time.

Each character can be one of the characters supported by the MPF-I/88. It takes a byte to represent a single character.

A memory space of 480 bytes in the system RAM is used as a display buffer so that MPF-I/88 supports a logical display screen of 20 columns by 24 rows. You can scroll the logical screen freely to view the desired portion of the logical display. Refer to MPF-I/88 User's Manual for how to scroll the display. In other words, with the buffer you are faciliated to see totally 24 rows of memory contents by pressing the ALT_A or the ALT_Z key.

There are 40 display positions on the physical LCD with each one has a physical address corresponding to it. However, each display position of the LCD is not addressable by the 8088.

The leftmost position of the first row is assigned with the address 80H, the rightmost of the first row is 93H, the leftmost of the second row is C0H, and the rightmost of the second row is D3H. We can view the 40 display positions on the LCD screen as memory locations separately ranging from 80H to 93H and from C0H to D3H.

The 8088 CPU can not directly access the 40 display positions on the LCD screen. Instead, it accesses the 40 display positions through four I/O ports in order to display and read characters on desired positions on the LCD. The four I/O port addresses are: 1A0H, 1A1H, 1A2H, and 1A3H.

Port 1A0H is used exclusively for receiving the write command from the CPU and transfering it to the LCD driver; port 1A1H is used for receiving data to be output the LCD and transfering it to the LCD driver. If you intend to know more about the functions of the LCD, please refer to the data sheet supplied by the LCD manufacturer.

Port 1A2 is used for receiving the read command from the CPU and tranfering it to the LCD driver; port 1A3 is used for receiving data to be input from the LCD and transfering it to the CPU.

Each LCD read or write operation involve many actions. For example, if you want to display a character on a certain display position, first you have to tell the CPU the display position you require; next, have the CPU check if the LCD is busy performing some operations; third, issue a display command through the CPU to the Command_Write I/O port; and finally transfer the data you want to display on the screen to the Data_Write I/O port. This holds true for reading data from the LCD screen.

The LCD device driver is identified by the procedure name OUT_LCD in MPF-I/88 Monitor Program Source Listing. You can refer to the procedure OUT_LCD in order to know how the LCD is driven. In order to find the OUT_LCD procedure, you must first refer to the cross reference section of the monitor source program to find the entry with OUT_LCD and then use the line number to locate the OUT_LCD procedure.

In order to let you trace the OUT_LCD procedure easier, an example program which is slightly different from the OUT_LCD procedure is provided as follows. Now let us look at our example program on LCD.

We will explain some of the frequently used assembler directives using examples in the example program.

```
                                                  PAGE     60,132
0000                           STACK              SEGMENT  PARA STACK 'STACK'
0000  0100 [                                      DB       256 DUP(20H)
                 20
                    ]

0100                           STACK              ENDS
                               ;
0000                           DATA               SEGMENT  PARA PUBLIC 'DATA'
                               ;
                               ;   I/O PORTS
                               ;
 = 01A0                        CMD_PORTW          EQU      01A0H
 = 01A1                        DATA_PORTW         EQU      01A1H
 = 01A2                        CMD_PORTR          EQU      01A2H
 = 01A3                        DATA_PORTR         EQU      01A3H
                               ;
                               ;   CONTROL CODE
                               ;
 = 0080                        ALT                =        80H
 = 0007                        BELL               =        07H
 = 000A                        LINEFEED           =        0AH
 = 000D                        RETURN             =        0DH
 = 000C                        FORMFEED           =        0CH
 = 0008                        BACKSPACE          =        08H
 = 00C4                        RIGHTARROW         =        44H+ALT               ;ALT-D
 = 00D3                        LEFTARROW          =        53H+ALT               ;ALT-S
 = 00C1                        UPCODE             =        41H+ALT               ;ALT-A
 = 00DA                        DOWNCODE           =        5AH+ALT               ;ALT-Z
                               ;
                               ;   CONSTANTS AND VARIABLES
                               ;
0000  14                       TWENTY             DB       20
0001  0016                     ROW                DW       22
0003  ??                       ADDRESSA           DB       ?
0004  ??                       ADDRESSB           DB       ?
0005  ??                       R_DATA             DB       ?
0006  80                       COUNT              DB       80H
0007  000D                     AREAAX             DW       000DH
0009  FFFF                     AREACX             DW       0FFFFH
000B  ??                       KEEP_CUR           DB       ?
000C  ??                       COL_VALUE          DB       ?
000D  ??                       COL_END            DB       ?
                               ;
                               ;   THE LCD BUFFER IN MEMORY  (20 COLUMNS BY 24 ROWS)
                               ;
000E      14 [                 ROW00              DB       20      DUP(0)
                 00
                    ]

0022      14 [                 ROW01              DB       20      DUP(0)
                 00
                    ]

0036      14 [                 ROW02              DB       20      DUP(0)
```

```
                    00
                         ]

      004A    14 [                    ROW03      DB     20     DUP(0)
                    00
                         ]

      005E    14 [                    ROW04      DB     20     DUP(0)
                    00
                         ]

      0072    14 [                    ROW05      DB     20     DUP(0)
                    00
                         ]

      0086    14 [                    ROW06      DB     20     DUP(0)
                    00
                         ]

      009A    14 [                    ROW07      DB     20     DUP(0)
                    00
                         ]

      00AE    14 [                    ROW08      DB     20     DUP(0)
                    00
                         ]

      00C2    14 [                    ROW09      DB     20     DUP(0)
                    00
                         ]

      00D6    14 [                    ROW10      DB     20     DUP(0)
                    00
                         ]

      00EA    14 [                    ROW11      DB     20     DUP(0)
                    00
                         ]

      00FE    14 [                    ROW12      DB     20     DUP(0)
                    00
                         ]

      0112    14 [                    ROW13      DB     20     DUP(0)
                    00
                         ]

      0126    14 [                    ROW14      DB     20     DUP(0)
                    00
                         ]

      013A    14 [                    ROW15      DB     20     DUP(0)
                    00
                         ]

```

```
014E      14 [                        ROW16         DB      20      DUP(Ø)
                  ØØ
                       ]

Ø162      14 [                        ROW17         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø176      14 [                        ROW18         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø18A      14 [                        ROW19         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø19E      14 [                        ROW2Ø         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø1B2      14 [                        ROW21         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø1C6      14 [                        ROW22         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø1DA      14 [                        ROW23         DB      2Ø      DUP(Ø)
                  ØØ
                       ]

Ø1EE                                  DATA     ENDS
                                      ;

                                      ;***********************************************************
                                      ; _____________LCD START _____________                      *
                                      ;***********************************************************
                                      ;   INPUT REQUIREMENT:
                                      ;
                                      ;   AL - CONTAINS THE ASCII CODE OF A CHARACTER TO BE OUTPUT
                                      ;   CH = Ø : MEANS THE CURSOR SHALL NOT BE PLACED ON THE SCREEN.
                                      ;   CH <> Ø: MEANS THE CURSOR SHALL BE PLACED ON THE SCREEN.
                                      ;   CL = Ø : CAUSE THE SCREEN NOT TO BE ABLE TO SCROLL.
                                      ;   CL <> Ø: PERFORM THE REVERSE OF "CL = Ø"
                                      ;
                                      ;-----------------------------------------------------------
ØØØØ                                  CODE     SEGMENT PARA PUBLIC 'CODE'
ØØØØ                                  OUT_LCD       PROC     FAR                 ;FUNCTION--> CONTRAL LCD
                                                    ASSUME   CS:CODE,DS:DATA
ØØØØ  E8 Ø346 R                                     CALL     PUSH_R              ;PUSH ALL THE REGISTERS

ØØØ3  BB ---- R                                     MOV      BX,DATA
ØØØ6  8E DB                                         MOV      DS,BX
```

```
0008  A3 0007 R                        MOV   AREAAX,AX          ;AREAAX = AX
000B  89 0E 0009 R                     MOV   AREACX,CX          ;AREACX = CX
000F  E8 032F R                        CALL  CUR_ONOFF          ;CURSOR ON OR OFF
                    ;::::::::::::::::::::::
                    ;CONTROL CODE TEST   ::
                    ;::::::::::::::::::::::
0012                FF:                                         ;FORMFEED
0012  3C 0C                            CMP   AL,FORMFEED        ; ?
0014  75 06                            JNZ   UDTEST             ;..
0016  E8 0088 R                        CALL  FF_SUB             ;ACTIVE
0019  EB 62 90                         JMP   RIGHT              ;OK!
001C                UDTEST:                                     ;TEST WINDOW POSITION
001C  83 3E 0001 R 16                  CMP   ROW,22             ;FIRST ROW = 22
0021  74 06                            JZ    BELL1
0023  E8 0304 R                        CALL  TEST_UD            ;TEST UP OR DOWN
0026  EB 55 90                         JMP   RIGHT
0029                BELL1:                                      ;BELL
0029  3C 07                            CMP   AL,BELL            ; ?
002B  75 05                            JNZ   BKSP               ;..
002D  CD 0C                            INT   0CH                ;ACTIVE
002F  EB 4C 90                         JMP   RIGHT              ;OK
0032                BKSP:                                       ;BACKSPACE
0032  3C 08                            CMP   AL,BACKSPACE       ; ?
0034  75 06                            JNZ   LF                 ;..
0036  E8 00B3 R                        CALL  BS_SUB             ;ACTIVE
0039  EB 42 90                         JMP   RIGHT              ;OK
003C                LF:                                         ;LINEFEED
003C  3C 0A                            CMP   AL,LINEFEED        ; ?
003E  75 06                            JNZ   RT                 ;..
0040  E8 00D2 R                        CALL  LF_SUB             ;ACTIVE
0043  EB 38 90                         JMP   RIGHT              ;OK
0046                RT:                                         ;RETURN
0046  3C 0D                            CMP   AL,RETURN          ; ?
0048  75 06                            JNZ   UU                 ;..
004A  E8 0136 R                        CALL  RT_SUB             ;ACTIVE
004D  EB 2E 90                         JMP   RIGHT              ;OK
0050                UU:                                         ;UPCODE
0050  3C C1                            CMP   AL,UPCODE          ; ?
0052  00 00 00 00                      JNZ   DD                 ;..
0056  E8 0272 R                        CALL  UP_LCD
0059  EB 22 90                         JMP   RIGHT
005C                DD:                                         ;DOWNCODE
005C  3C DA                            CMP   AL,DOWNCODE        ; ?
005E  75 06                            JNZ   RA                 ;..
0060  E8 029B R                        CALL  DOWN_LCD
0063  EB 18 90                         JMP   RIGHT              ;OK
0066                RA:                                         ;RIGHT
0066  3C C4                            CMP   AL,RIGHTARROW      ; ?
0068  75 06                            JNZ   LA                 ;..
006A  E8 014B R                        CALL  RA_SUB             ;ACTIVE
006D  EB 0E 90                         JMP   RIGHT              ;OK
0070                LA:                                         ;LEFTARROW
0070  3C D3                            CMP   AL,LEFTARROW       ; ?
0072  75 06                            JNZ   DISPLAY            ;DISPLAY
0074  E8 0176 R                        CALL  LA_SUB             ;ACTIVE
```

```
0077  EB 04 90                               JMP    RIGHT           ;OK
007A                        DISPLAY:                                ;DISPLAY
007A  E8 017A R                              CALL   DISP_SUB        ;ACTIVE
007D                        RIGHT:                                  ;.......
007D  A1 0007 R                              MOV    AX,AREAAX       ;RESTORE AX
0080  8B 0E 0009 R                           MOV    CX,AREACX       ;RESTORE CX
0084  E8 0359 R                              CALL   POP_R           ;POP
0087  CB                                     RET                    ;............
0088                        OUT_LCD          ENDP                   ;............
                            ;-----------------------------------------------------
                            ;     __________  FORMFEED SUBROUTINE__________      :
                            ;-----------------------------------------------------
0088                        FF_SUB           PROC   NEAR            ;FUNCTION FOREFEED
0088  B0 38                                  MOV    AL,38H          ;RESET CODE
008A  E8 01D0 R                              CALL   OUT_FUN         ;TWICE
008D  E8 01D0 R                              CALL   OUT_FUN         ;FUNCTION SET TWICE
0090  B0 0D                                  MOV    AL,0DH          ;SET ON DISPLAY AND BLINK
0092  E8 01D0 R                              CALL   OUT_FUN         ;ACTIVE
0095  B0 06                                  MOV    AL,6            ;SET CURSOR MOVE DIRECTIVE (RIGHT)
0097  E8 01D0 R                              CALL   OUT_FUN         ;ACTIVE
009A  B0 01                                  MOV    AL,1            ;CLEAR DISPLAY, CURSOR TO HOME
009C  E8 01D0 R                              CALL   OUT_FUN         ;ACTIVE
009F  C6 06 0006 R 80                        MOV    COUNT,80H       ;INITIAL
00A4  C7 06 0001 R 0016                      MOV    ROW,22          ;INITIALIZE ROW TO 20
00AA  C6 06 0000 R 14                        MOV    TWENTY,20       ;TWENTY EQUALS TO 20
00AF  E8 031E R                              CALL   CLRTAB          ;CLEAR LCD TABLE
00B2  C3                                     RET                    ;.................
00B3                        FF_SUB           ENDP                   ;.................
                            ;-----------------------------------------------------
                            ;                BACKSPACE SUBROUTINE                 |
                            ;-----------------------------------------------------
00B3                        BS_SUB           PROC   NEAR            ;FUNCTION--> BACK SPACE
00B3  A0 0006 R                              MOV    AL,COUNT        ;CURSOR IN ADDRESS 0 (ROW 1)
00B6  3C 80                                  CMP    AL,80H          ; ?
00B8  74 17                                  JZ     SUBRIGHT        ;OK
00BA  3C C0                                  CMP    AL,0C0H         ;CURSOR IN ADDRESS 21 (ROW 2)
00BC  74 06                                  JZ     BKSPB           ;......
00BE  E8 01B4 R                              CALL   BACKSP          ;ACTIVE
00C1  EB 0E 90                               JMP    SUBRIGHT        ;OK
00C4  B0 94                 BKSPB:           MOV    AL,94H          ;CURSOR TO ADDRESS 20
00C6  E8 01D0 R                              CALL   OUT_FUN         ;ACTIVE
00C9  E8 01B4 R                              CALL   BACKSP          ;BACK SPACE
00CC  C6 06 0006 R 93                        MOV    COUNT,93H       ;ON ROW1 COL20
00D1                        SUBRIGHT:                               ;OK
00D1  C3                                     RET                    ;.................
00D2                        BS_SUB           ENDP                   ;.................
                            ;-----------------------------------------------------
                            ;                                                     |
                            ;                LINEFEED SUBROUTINE                  |
                            ;                                                     |
                            ;-----------------------------------------------------
00D2                        LF_SUB           PROC   NEAR            ;LINE FEED
00D2  A0 0006 R                              MOV    AL,COUNT        ;CURSOR IN ROW(1) OR ROW(2)
00D5  3C C0                                  CMP    AL,0C0H         ;ROW2 ?
00D7  7C 33                                  JL     CLR_ROW2        ;CURSOR IN ROW(1)
```

```
00D9  E8 01F9 R                          CALL   SCROLLER           ;SCROLL TABLE
00DC  C6 06 0003 R 80      LF_ROW2:      MOV    ADDRESSA,80H       ;CURSOR IN ROW(2)
00E1  C6 06 0004 R C0                    MOV    ADDRESSB,0C0H      ;ROW2 COL1
00E6  A0 0004 R            NEX_DATA:     MOV    AL,ADDRESSB        ;......................
00E9  3C D4                              CMP    AL,0D4H            ;ROW2 COL20
00EB  74 1F                              JZ     CLR_ROW2           ;CLEAR ROW 2
00ED  E8 01D0 R                          CALL   OUT_FUN            ;ACTIVE
00F0  E8 01F3 R                          CALL   IN_DATA            ;INPUT DATA
00F3  A2 0005 R                          MOV    R_DATA,AL          ;R_DATA<--AL
00F6  A0 0003 R                          MOV    AL,ADDRESSA        ;ON LCD ADDRESS
00F9  E8 01D0 R                          CALL   OUT_FUN            ;ACTIVE
00FC  A0 0005 R                          MOV    AL,R_DATA          ;AL = R_DATA
00FF  E8 01ED R                          CALL   OUT_VAL            ;ACTIVE
0102  FE 06 0003 R                       INC    ADDRESSA           ;MOVE CURSOR POSITION
0106  FE 06 0004 R                       INC    ADDRESSB           ;....................
010A  EB DA                              JMP    NEX_DATA           ;NEXT DATA
010C  C6 06 0004 R C0      CLR_ROW2:     MOV    ADDRESSB,0C0H      ;ROW2 LCD LOCATION
0111  A0 0004 R            CLR_SPA:      MOV    AL,ADDRESSB        ;.................
0114  3C D4                              CMP    AL,0D4H            ;END ROW 2 ?
0116  74 0E                              JZ     OUT_POSITION       ;...........
0118  E8 01D0 R                          CALL   OUT_FUN            ;ACTIVE
011B  B0 20                              MOV    AL,20H             ;SHOW " "
011D  E8 01ED R                          CALL   OUT_VAL            ;ACTIVE
0120  FE 06 0004 R                       INC    ADDRESSB           ;NEXT LOCATION
0124  EB EB                              JMP    CLR_SPA            ;CLEAR SPACE
0126                       OUT_POSITION:                           ;OUT POSITION
0126  A0 0006 R                          MOV    AL,COUNT           ;COMPARE CURSOR POSITION
0129  3C C0                              CMP    AL,0C0H            ;ON ROW2 COL1 ?
012B  7D 05                              JGE    NO_CHANG           ;..............
012D  04 40                              ADD    AL,40H             ;CURSOR ON ROW1
012F  A2 0006 R                          MOV    COUNT,AL           ;..............
0132  E8 01D0 R            NO_CHANG:     CALL   OUT_FUN            ;ACTIVE
0135  C3                                 RET                       ;.............................
0136                       LF_SUB        ENDP                      ;.............................
                           ;---------------------------------------------------------
                           ;                                                        |
                           ;             RETURN SUBROUTINE                          |
                           ;                                                        |
                           ;---------------------------------------------------------
0136                       RT_SUB        PROC   NEAR               ;FUNCTION --> RETURN CURSOR
0136  80 3E 0006 R C0                    CMP    COUNT,0C0H         ;COMPARE ROW ?
013B  7D 05                              JGE    RT_ROW2            ;.............
013D  B0 80                              MOV    AL,80H             ;RETURN ROW 1  COL 1
013F  EB 03 90                           JMP    RT_ROW1            ;...................
0142  B0 C0                RT_ROW2:      MOV    AL,0C0H            ;0C0H IS ROW2 COL1 LOCATION
0144  A2 0006 R            RT_ROW1:      MOV    COUNT,AL           ;..........................
0147  E8 01D0 R                          CALL   OUT_FUN            ;ACTIVE
014A  C3                                 RET                       ;..........................
014B                       RT_SUB        ENDP                      ;..........................
                           ;---------------------------------------------------------
                           ;                                                        |
                           ;             RIGHTARROW SUBROUTINE                      |
                           ;                                                        |
                           ;---------------------------------------------------------
014B                       RA_SUB        PROC   NEAR               ;FUNCTION --> MOVE CURSOR TO RIGHT
```

```
Ø14B  8Ø 3E ØØØ6 R D3                   CMP     COUNT,ØD3H      ;CURSOR ON ROW2 COL2Ø ?
Ø15Ø  74 1B                             JZ      NEX_ROW         ;......................
Ø152  8Ø 3E ØØØ6 R 93                   CMP     COUNT,93H       ;CURSOR ON ROW1 COL2Ø ?
Ø157  75 Ø8                             JNZ     RA_CTN          ;......................
Ø159  BØ CØ                             MOV     AL,ØCØH         ;MOVE CURSOR ROW2 COL1
Ø15B  A2 ØØØ6 R                         MOV     COUNT,AL        ;.....................
Ø15E  EB Ø7 9Ø                          JMP     RA_ROW2         ;.....................
Ø161                      RA_CTN:                               ;RIGHT CONTINUE
Ø161  BØ 14                             MOV     AL,14H          ;LCD TO RIGHT FUNCTION
Ø163  FE Ø6 ØØØ6 R                      INC     COUNT           ;....................
Ø167                      RA_ROW2:                              ;ON ROW2 TO RIGHT
Ø167  E8 Ø1DØ R                         CALL    OUT_FUN         ;ACTIVE
Ø16A  EB Ø9 9Ø                          JMP     RA_RET          ;OK
Ø16D                      NEX_ROW:                              ;NEXT ROW
Ø16D  C6 Ø6 ØØØ6 R CØ                   MOV     COUNT,ØCØH      ;ROW2
Ø172  E8 ØØD2 R                         CALL    LF_SUB          ;LINE FEED
Ø175                      RA_RET:                               ;RETURN
Ø175  C3                                RET                     ;............................
Ø176                      RA_SUB        ENDP                    ;............................
                          ;-------------------------------------------------------------
                          ;                                                            |
                          ;             LEFTARROW SUBROUTINE                           |
                          ;                                                            |
                          ;-------------------------------------------------------------
Ø176                      LA_SUB        PROC    NEAR            ;FUNCTION --> TO LEFT
Ø176  E8 ØØB3 R                         CALL    BS_SUB          ;EQUIMENT BACK SPACE
Ø179  C3                                RET                     ;............................
Ø17A                      LA_SUB        ENDP                    ;............................
                          ;-------------------------------------------------------------
                          ;                                                            |
                          ;             DISPLAY CHARACTER TO LCD                       |
                          ;                                                            |
                          ;-------------------------------------------------------------
Ø17A                      DISP_SUB      PROC    NEAR            ;FUNCTION --> DISPLAY
Ø17A  E8 Ø1ED R                         CALL    OUT_VAL         ;ACTIVE
Ø17D  FE Ø6 ØØØ6 R                      INC     COUNT           ;...........
Ø181  AØ ØØØ6 R                         MOV     AL,COUNT        ;.............
Ø184  3C D4                             CMP     AL,ØD4H         ;IF CURSOR OVER ROW<2>
Ø186  74 ØF                             JZ      DISPSCROLL      ;SCROLL
Ø188  3C 94                             CMP     AL,Ø94H         ;IF CURSOR OVER ROW<1>
Ø18A  75 ØE                             JNZ     DISPRIGHT       ;...................
Ø18C  BØ CØ                             MOV     AL,ØCØH         ;SET CURSOR ON ROW2 COL1
Ø18E  A2 ØØØ6 R                         MOV     COUNT,AL        ;.......................
Ø191  E8 Ø1DØ R                         CALL    OUT_FUN         ;ACTIVE
Ø194  EB Ø4 9Ø                          JMP     DISPRIGHT       ;OK
Ø197                      DISPSCROLL:                           ;DISPLAY SCROLL
Ø197  E8 Ø19B R                         CALL    SCROLL          ;ACTIVE
Ø19A                      DISPRIGHT:                            ;OK
Ø19A  C3                                RET                     ;............................
Ø19B                      DISP_SUB      ENDP                    ;............................
                          ;-------------------------------------------------------------
                          ;                                                            |
                          ;             IF CL=Ø THEN NO SCROLL                         |
                          ;              NOT =Ø THEN SCROLL                            |
                          ;                                                            |
                                  ;----------------------------------------------------------
     Ø19B                         SCROLL       PROC     NEAR             ;FUNCTION --> SCROLL LCD
     Ø19B  8Ø F9 ØØ                            CMP      CL,Ø             ;CL = Ø ?
     Ø19E  75 ØB                               JNZ      ASCROLL          ;.........
     Ø1AØ  BØ D3                               MOV      AL,ØD3H          ;DONOT SCROLL
     Ø1A2  A2 ØØØ6 R                           MOV      COUNT,AL         ;............
     Ø1A5  E8 Ø1DØ R                           CALL     OUT_FUN          ;ACTIVE
     Ø1A8  EB Ø9 9Ø                            JMP      SCRRIGHT         ;OK
     Ø1AB                         ASCROLL:                               ;................
     Ø1AB  C6 Ø6 ØØØ6 R CØ                     MOV      COUNT,ØCØH       ;CL NOT EQU Ø ,S_FLAG
     Ø1BØ  E8 ØØD2 R                           CALL     LF_SUB           ;LINE FEED
     Ø1B3                         SCRRIGHT:                              ;SCROLL RIGHT
     Ø1B3  C3                                  RET                       ;...................
     Ø1B4                         SCROLL       ENDP                      ;............................
                                  ;;--------------------------------------------------------
                                  ;;           BACKSPACE ROUTINE                            |
                                  ;;---------- SUBROUTINE  ----------                      |
                                  ;;                                                      |
                                  ;;--------------------------------------------------------
     Ø1B4                         BACKSP       PROC  NEAR                ;FUNCTION --> BACK SPACE
     Ø1B4  BØ 1Ø                               MOV      AL,1ØH           ;CUROR SHIFT LEFT
     Ø1B6  E8 Ø1DØ R                           CALL     OUT_FUN          ;ACTIVE
     Ø1B9  A1 ØØØ7 R                           MOV      AX,AREAAX        ;GET CHARACTER VALUE
     Ø1BC  3C D3                               CMP      AL,LEFTARROW     ;COMPARE LEFTAROW CODE ?
     Ø1BE  74 ØA                               JZ       FINISH           ;OK
     Ø1CØ  BØ 2Ø                               MOV      AL,2ØH           ;2ØH = ' '
     Ø1C2  E8 Ø1ED R                           CALL     OUT_VAL          ;ACTIVE
     Ø1C5  BØ 1Ø                               MOV      AL,1ØH           ;LCD CURSOR TO LEFT
     Ø1C7  E8 Ø1DØ R                           CALL     OUT_FUN          ;ACTIVE
     Ø1CA                         FINISH:                                ;.................
     Ø1CA  8Ø 2E ØØØ6 R Ø1                     SUB      COUNT,1          ;.................
     Ø1CF  C3                                  RET                       ;RETURN......................
     Ø1DØ                         BACKSP       ENDP                      ;............................
                                  ;----------------------------------------------------------
                                  ;____________ OUT_FUNCTION ____________              :
                                  ;==========================================================
     Ø1DØ                         OUT_FUN      PROC     NEAR             ;FUNCTION --> DO LCD FUNCTION
     Ø1DØ  52                                  PUSH     DX               ;.................
     Ø1D1  BA Ø1AØ                             MOV      DX,CMD_PORTW     ;CMD_PORTW = 1AØH
     Ø1D4                         OUTA:                                  ;...............
     Ø1D4  52                                  PUSH     DX               ;PUSH REGIST
     Ø1D5  5Ø                                  PUSH     AX               ;.............
     Ø1D6  BA Ø1A2                             MOV      DX,CMD_PORTR     ;CMD_PORTR = 1A2H
     Ø1D9  EC                     WAIT:        IN       AL,DX            ;DX = PORT
     Ø1DA  ØA CØ                               OR       AL,AL            ;SET (SF)=1
     Ø1DC  78 FB                               JS       WAIT             ;LCD BUSY ?
     Ø1DE  58                                  POP      AX               ;...........
     Ø1DF  5A                                  POP      DX               ;............
     Ø1EØ  81 FA Ø1A3                          CMP      DX,DATA_PORTR    ;DATA_PORTR = 1A3H
     Ø1E4  74 Ø4                               JZ       READ_DATA        ;................
     Ø1E6  EE                                  OUT      DX,AL            ;ACTIVE
     Ø1E7  EB Ø2 9Ø                            JMP      FINE             ;OK
     Ø1EA                         READ_DATA:                             ;READ DATA
     Ø1EA  EC                                  IN       AL,DX            ;ACTIVE
     Ø1EB                         FINE:                                  ;........
```

```
 Ø1EB  5A                                    POP     DX                ;........
 Ø1EC  C3                                    RET                       ;...........................
 Ø1ED                        OUT_FUN         ENDP                      ;...........................
                             ;-------------------------------------------------
                             ;               OUT VALUE                       :
                             ;               OUT_VAL                         :
                             ;-------------------------------------------------
 Ø1ED                        OUT_VAL         PROC    NEAR              ;FUNCTION --> OUT CHARACTER LCD
 Ø1ED  52                                    PUSH    DX                ;...............
 Ø1EE  BA Ø1A1                               MOV     DX,DATA_PORTW     ;DATA_PORTW = 1A1H
 Ø1F1  EB E1                                 JMP     OUTA              ;........
 Ø1F3                        OUT_VAL         ENDP                      ;............................
                             ;-------------------------------------------------
                             ;____________    READ DATA  ______________   :
                             ;-------------------------------------------------
 Ø1F3                        IN_DATA         PROC    NEAR              ;FUNCTION --> READ DATA
 Ø1F3  52                                    PUSH    DX                ;.............
 Ø1F4  BA Ø1A3                               MOV     DX,DATA_PORTR     ;DATA_PORTR = 1A3H
 Ø1F7  EB DB                                 JMP     OUTA              ;...........
 Ø1F9                        IN_DATA         ENDP                      ;............................
                             ;-------------------------------------------------
                             ;               SCROLL TABLE UP ONE LOW         :
                             ;               SCROLLER                        :
                             ;-------------------------------------------------
 Ø1F9                        SCROLLER        PROC    NEAR              ;FUNCTION --> SCROLL TABLE
 Ø1F9  83 3E ØØØ1 R 16                       CMP     ROW,22            ;ROW = 22 ?
 Ø1FE  75 1E                                 JNE     SCRIGHT           ; ROW (22) IS START ADDRESS
 Ø2ØØ  5Ø                                    PUSH    AX                ;PUSH
 Ø2Ø1  51                                    PUSH    CX                ;....
 Ø2Ø2  57                                    PUSH    DI                ;....
 Ø2Ø3  56                                    PUSH    SI                ;....
 Ø2Ø4  Ø6                                    PUSH    ES                ;....
                                             ASSUME  ES:DATA           ;ES -->DATA SEGMENT
 Ø2Ø5  B8 ---- R                             MOV     AX,DATA           ; AX = DATA SEGMENT
 Ø2Ø8  8E CØ                                 MOV     ES,AX             ; ES =>DATA SEGMENT
 Ø2ØA  BF ØØØE R                             MOV     DI,OFFSET ROWØØ   ; ROWØØ ADDRESS
 Ø2ØD  BE ØØ22 R                             MOV     SI,OFFSET ROWØ1   ; ES:[DI] <== DS[SI]
 Ø21Ø  B9 Ø1CC                               MOV     CX,46Ø            ; MOVE 46Ø BYTES 2Ø*23
 Ø213  FC                                    CLD                       ; (DF)=Ø
 Ø214  F3/ A4                                REP     MOVSB             ;
 Ø216  Ø7                                    POP     ES                ;POP REGISTER
 Ø217  5E                                    POP     SI                ;...........
 Ø218  5F                                    POP     DI                ;...........
 Ø219  59                                    POP     CX                ;...........
 Ø21A  58                                    POP     AX                ;...........
 Ø21B  E8 Ø21F R                             CALL    LCD_TABLE         ; COPY LCD ROW TO TABLE
 Ø21E                        SCRIGHT:                                  ; OK
 Ø21E  C3                                    RET                       ;...........................
 Ø21F                        SCROLLER        ENDP                      ;...........................
                             ;-------------------------------------------------
                             ;               MOV LCD ROW(Ø) TO TABLE ROW(21) :
                             ;               LCD_TABLE                       :
                             ;-------------------------------------------------
 Ø21F                        LCD_TABLE       PROC    NEAR              ;FUNCTION --> MOVE LCD TO TABLE
 Ø21F  56                                    PUSH    SI                ;........
```

```
      0220  BE 01B2 R                         MOV     SI,OFFSET ROW21       ; TABLE ADDRESS ROW21
      0223  C6 06 000C R 80                   MOV     COL_VALUE,80H         ; START ADDRESS
      0228  C6 06 000D R 94                   MOV     COL_END,094H          ; INPUT LCD ADDRESS
      022D  E8 0232 R                         CALL    COPYROW               ; COPY ONE ROW
      0230                      SRIGHT:                                     ; OK
      0230  5E                                POP     SI                    ;............
      0231  C3                                RET                           ;...........................
      0232                      LCD_TABLE     ENDP                          ;...........................
                                ;------------------------------------------------------
                                ;             COPY ONE ROW TO TABLE                    :
                                ;             COPYROW                                  :
                                ;------------------------------------------------------
      0232                      COPYROW       PROC    NEAR                  ;INPUT => SI
      0232  50                                PUSH    AX                    ;          COL_VALUE
      0233  56                                PUSH    SI                    ;          COL_END
      0234                      SCR_CON:                                    ;..........
      0234  A0 000C R                         MOV     AL,COL_VALUE          ;ADDRESS LCD
      0237  3A 06 000D R                      CMP     AL,COL_END            ;LCD END ADDRESS
      023B  74 0F                             JZ      CRIGHT                ;OK
      023D  E8 01D0 R                         CALL    OUT_FUN               ;IN DATA TO TABLE
      0240  E8 01F3 R                         CALL    IN_DATA               ;ACTIVE
      0243  88 04                             MOV     [SI],AL               ;MOVE CHARACTER TO TABLE
      0245  FE 06 000C R                      INC     COL_VALUE             ;.............
      0249  46                                INC     SI                    ;.............
      024A  EB E8                             JMP     SCR_CON               ;GET NEXT VALUE
      024C                      CRIGHT:                                     ;OK
      024C  5E                                POP     SI                    ;POP REGISTER
      024D  58                                POP     AX                    ;.............
      024E  C3                                RET                           ;...........................
      024F                      COPYROW       ENDP                          ;...........................
                                ;------------------------------------------------------
                                ;             COPY LCD TWO ROW TO TABLE ROW(22,23)     :
                                ;             COPYLCD                                  :
                                ;------------------------------------------------------
      024F                      COPYLCD       PROC    NEAR                  ;FUNCTION --> COPY LCD TO TABLE
      024F  56                                PUSH    SI                    ;
      0250  BE 01C6 R                         MOV     SI,OFFSET ROW22       ;ROW22 ADDRESS
      0253  C6 06 000C R 80                   MOV     COL_VALUE,80H         ;COPY LCD ROW(0)
      0258  C6 06 000D R 94                   MOV     COL_END,94H           ;COL END
      025D  E8 0232 R                         CALL    COPYROW               ;COPY ROW1
      0260  BE 01DA R                         MOV     SI,OFFSET ROW23       ;COPY ROW2
      0263  C6 06 000C R C0                   MOV     COL_VALUE,0C0H        ;COPY LCD ROW(1)
      0268  C6 06 000D R D4                   MOV     COL_END,0D4H          ;COL END
      026D  E8 0232 R                         CALL    COPYROW               ;ACTIVE
      0270  5E                                POP     SI                    ;.......
      0271  C3                                RET                           ;...........................
      0272                      COPYLCD       ENDP                          ;...........................
                                ;------------------------------------------------------
                                ;             UP LCD                                   :
                                ;             UP_LCD                                   :
                                ;------------------------------------------------------
      0272                      UP_LCD        PROC    NEAR                  ;FUNCTION —> TO UP
      0272  51                                PUSH    CX                    ;..........
      0273  B9 0000                           MOV     CX,0                  ;CX = 0
      0276  E8 032F R                         CALL    CUR_ONOFF             ;OFF CURSOR
```

```
Ø279  83 3E ØØØ1 R ØØ              CMP    ROW,Ø                 ;ROW = Ø ?
Ø27E  74 19                        JZ     URIGHT                ;CAN NOT UP
Ø28Ø  83 3E ØØØ1 R 16              CMP    ROW,22                ;TABLE BUTTON ?
Ø285  75 ØB                        JNZ    NONCOPY               ;NOT IN CORRECT POSITION
Ø287  8A ØE ØØØ6 R                 MOV    CL,COUNT              ;CL = CURSOR POSITION
Ø28B  88 ØE ØØØB R                 MOV    KEEP_CUR,CL           ;..........
Ø28F  E8 Ø24F R                    CALL   COPYLCD               ;DO COPY
Ø292                    NONCOPY:                                ;NO COPY
Ø292  FF ØE ØØØ1 R                 DEC    ROW                   ;ROW - 1
Ø296  E8 Ø2D9 R                    CALL   MOVLCD                ;UP LCD
Ø299                    URIGHT:                                 ;OK
Ø299  59                           POP    CX                    ;....
Ø29A  C3                           RET                          ;.............................
Ø29B                    UP_LCD     ENDP                         ;.............................
                        ;--------------------------------------------------------
                        ;          DOWN LCD                              :
                        ;          DOWN_LCD                              :
                        ;--------------------------------------------------------
Ø29B                    DOWN_LCD   PROC   NEAR                  ;FUNCTION --> DOWN LCD
Ø29B  5Ø                           PUSH   AX                    ;.....
Ø29C  51                           PUSH   CX                    ;
Ø29D  83 3E ØØØ1 R 16              CMP    ROW,22                ;ROW = 22?
Ø2A2  74 14                        JZ     DRIGHT                ;IN BUTTON
Ø2A4  FF Ø6 ØØØ1 R                 INC    ROW                   ;...............
Ø2A8  83 3E ØØØ1 R 16              CMP    ROW,22                ;IF RETURN CORRECT POSITION
Ø2AD  75 Ø6                        JNE    NOMAL                 ;...............
Ø2AF  E8 Ø2BB R                    CALL   RES_LCD               ;RETURN CORRECT POSITION
Ø2B2  EB Ø4 9Ø                     JMP    DRIGHT                ;OK
Ø2B5                    NOMAL:                                  ;NOMAL
Ø2B5  E8 Ø2D9 R                    CALL   MOVLCD                ;DOWN LCD
Ø2B8                    DRIGHT:                                 ;OK
Ø2B8  59                           POP    CX                    ;
Ø2B9  58                           POP    AX                    ;
Ø2BA  C3                           RET                          ;.............................
Ø2BB                    DOWN_LCD   ENDP                         ;.............................
                        ;--------------------------------------------------------
                        ;          RESET LCD POSITION                    :
                        ;          RES_LCD                               :
                        ;--------------------------------------------------------
Ø2BB                    RES_LCD    PROC   NEAR                  ;FUNCTION --> RETURN LCD POSITION
Ø2BB  5Ø                           PUSH   AX                    ;...................
Ø2BC  51                           PUSH   CX                    ;...................
Ø2BD  C7 Ø6 ØØØ1 R ØØ16            MOV    ROW,22                ;ROW = 22
Ø2C3  E8 Ø2D9 R                    CALL   MOVLCD                ;COPY TABLE ROW(22,23) TO LCD
Ø2C6  8B ØE ØØØ9 R                 MOV    CX,AREACX             ;ON CURSOR
Ø2CA  E8 Ø32F R                    CALL   CUR_ONOFF             ;CURSOR ON OR OFF
Ø2CD  AØ ØØØB R                    MOV    AL,KEEP_CUR           ;SET CURSOR
Ø2DØ  A2 ØØØ6 R                    MOV    COUNT,AL              ;..........
Ø2D3  E8 Ø1DØ R                    CALL   OUT_FUN               ;ACTIVE
Ø2D6  59                           POP    CX                    ;..............
Ø2D7  58                           POP    AX                    ;..............
Ø2D8  C3                           RET                          ;..............................
Ø2D9                    RES_LCD    ENDP                         ;..............................
                        ;--------------------------------------------------------
                        ;          MOVE LCD UP OR DOWN                   :
                                ;               MOVLCD                                  :
                                ;--------------------------------------------------------
0Ø2D9                           MOVLCD          PROC    NEAR            ;FUNCTION --> MOVE LCD UP OR DOWN
Ø2D9   5Ø                                       PUSH    AX              ;............
Ø2DA   51                                       PUSH    CX              ;.........
Ø2DB   56                                       PUSH    SI              ;...........
Ø2DC   BØ 8Ø                                    MOV     AL,Ø8ØH         ;CURSOR ON ROW1 COL1
Ø2DE   A2 ØØØ6 R                                MOV     COUNT,AL        ;SET COUNT
Ø2E1   E8 Ø1DØ R                                CALL    OUT_FUN         ;SET CURSOR IN START
Ø2E4   BE ØØØE R                                MOV     SI,OFFSET ROWØØ ;ROWØØ ADDRESS
Ø2E7   A1 ØØØ1 R                                MOV     AX,ROW          ;ROWØØ+ROW*2Ø
Ø2EA   8A ØE ØØØØ R                             MOV     CL,TWENTY       ;............
Ø2EE   F6 E1                                    MUL     CL              ;GET ADDRESS
Ø2FØ   Ø3 FØ                                    ADD     SI,AX           ;............
Ø2F2   B9 ØØ28                                  MOV     CX,4Ø           ;4Ø TIMES
Ø2F5                            MOVCHAR:                                ;MOVE CHARACTER
Ø2F5   51                                       PUSH    CX              ;......
Ø2F6   FC                                       CLD                     ;(DF)=Ø
Ø2F7   AC                                       LODSB                   ;[SI] => AL
Ø2F8   B1 ØØ                                    MOV     CL,Ø            ;OFF SCROLL
Ø2FA   E8 Ø17A R                                CALL    DISP_SUB        ;DISPLAY A CHARACTER
Ø2FD   59                                       POP     CX              ;.........
Ø2FE   E2 F5                                    LOOP    MOVCHAR         ;..............
Ø3ØØ   5E                                       POP     SI              ;.........
Ø3Ø1   59                                       POP     CX              ;
Ø3Ø2   58                                       POP     AX              ;
Ø3Ø3   C3                                       RET                     ;...........................
Ø3Ø4                            MOVLCD          ENDP                    ;...........................
                                ;--------------------------------------------------------
                                ;               TEST UP DOWN CONDITION                  :
                                ;               TEST_UD                                 :
                                ;--------------------------------------------------------
Ø3Ø4                            TEST_UD         PROC    NEAR            ;FUNCTION --> TEST ROW CONDITION
Ø3Ø4   5Ø                                       PUSH    AX              ;
Ø3Ø5   3C C1                                    CMP     AL,UPCODE       ;UPCODE ?
Ø3Ø7   75 Ø6                                    JNE     CMPDOWN         ;IF DOWNCODE CODE
Ø3Ø9   E8 Ø272 R                                CALL    UP_LCD          ;ACTIVE
Ø3ØC   EB ØE 9Ø                                 JMP     TUDRIGHT        ;OK
Ø3ØF                            CMPDOWN:                                ;COMPARE DOWN
Ø3ØF   3C DA                                    CMP     AL,DOWNCODE     ;DOWNCODE ?
Ø311   75 Ø6                                    JNE     RES_UD          ;....
Ø313   E8 Ø29B R                                CALL    DOWN_LCD
Ø316   EB Ø4 9Ø                                 JMP     TUDRIGHT        ;OK
Ø319                            RES_UD:                                 ;RESET
Ø319   E8 Ø2BB R                                CALL    RES_LCD         ;ACTIVE
Ø31C                            TUDRIGHT:                               ;RIGHT
Ø31C   58                                       POP     AX              ;....
Ø31D   C3                                       RET                     ;...........................
Ø31E                            TEST_UD         ENDP                    ;...........................
                                ;--------------------------------------------------------
                                ;               CLEAR LCD TABLE                         :
                                ;               CLRTAB                                  :
                                ;--------------------------------------------------------
Ø31E                            CLRTAB          PROC    NEAR            ;FUNCTION --> CLEAR TABLE
Ø31E   56                                       PUSH    SI              ;PUSH REGISTER
```

```
031F  51                         PUSH    CX                   ;.............
0320  BE 000E R                  MOV     SI,OFFSET ROW00      ;ROW00 ADDRESS
0323  B9 01B8                    MOV     CX,440               ;20 * 22
0326                 TABLP:                                   ;LOOP
0326  C6 04 20                   MOV     BYTE PTR [SI],20H    ;ASCII CODE 20H = SPACE
0329  46                         INC     SI                   ;SI + 1
032A  E2 FA                      LOOP    TABLP                ;LOOP
032C  59                         POP     CX                   ;POP REGISTER
032D  5E                         POP     SI                   ;............
032E  C3                         RET                          ;......................................
032F                 CLRTAB      ENDP                         ;......................................
                     ;::::::::::::::::::::::::::::::::::::::::::
                     ;:                                      ::
                     ;:       CURSOR ON OR OFF               ::
                     ;:                                      ::
                     ;::::::::::::::::::::::::::::::::::::::::::
032F                 CUR_ONOFF   PROC    NEAR                 ;INPUT CX =  0  CURSOR OFF
032F  50                         PUSH    AX                   ;           =/ 0  CURSOR ON
0330  51                         PUSH    CX                   ;
0331  80 FD 00                   CMP     CH,0                 ;CH=0 IS NO CURSOR
0334  75 08                      JNZ     CUR_ON               ;.................
0336                 CUR_OFF:                                 ;CURSOR OFF
0336  B0 0C                      MOV     AL,0CH               ;CURSOR OFF
0338  E8 01D0 R                  CALL    OUT_FUN              ;ACTIVE
033B  EB 06 90                   JMP     CURRET               ;RETURN
033E                 CUR_ON:                                  ;CURSOR ON
033E  B0 0D                      MOV     AL,0DH               ;CURSOR ON
0340  E8 01D0 R                  CALL    OUT_FUN              ;ACTIVE
0343                 CURRET:                                  ;RIGHT
0343  59                         POP     CX                   ;POP REGISTER
0344  58                         POP     AX                   ;............
0345  C3                         RET                          ;RETURN
0346                 CUR_ONOFF   ENDP                         ;END
                     ;;::::::::::::::::::::::::::::::::::::::::::
                     ;;                                       ::
                     ;;       REGISTERS PUSHING & POPING      ::
                     ;;              ROUTINES                 ::
                     ;;                                       ::
                     ;;::::::::::::::::::::::::::::::::::::::::::
0346                 PUSH_R  PROC    NEAR                     ;PUSH ALL REGISTER
0346  2E: 8F 06 036C R       POP     CS:IP_MEM
034B  50                     PUSH    AX
034C  53                     PUSH    BX
034D  51                     PUSH    CX
034E  52                     PUSH    DX
034F  57                     PUSH    DI
0350  56                     PUSH    SI
0351  1E                     PUSH    DS
0352  06                     PUSH    ES
0353  2E: FF 36 036C R       PUSH    CS:IP_MEM
0358  C3                     RET
0359                 PUSH_R  ENDP

0359                 POP_R   PROC    NEAR                     ;POP ALL REGISTER
0359  2E: 8F 06 036C R       POP     CS:IP_MEM
```

```
ØØ35E  Ø7                                  POP     ES
Ø35F  1F                                  POP     DS
Ø360  5E                                  POP     SI
Ø361  5F                                  POP     DI
Ø362  5A                                  POP     DX
Ø363  59                                  POP     CX
Ø364  5B                                  POP     BX
Ø365  58                                  POP     AX
Ø366  2E: FF 36 Ø36C R                    PUSH    CS:IP_MEM
Ø36B  C3                                  RET
Ø36C                             POP_R    ENDP
                                 ;
Ø36C  ØØØØ                       IP_MEM           DW      Ø
                                 ;
Ø36E                             CODE     ENDS
                                          END     OUT_LCD
```

```
Segments and groups:

                N a m e                 Size    align   combine class

CODE . . . . . . . . . . . . . .        036E    PARA    PUBLIC  'CODE'
DATA . . . . . . . . . . . . . .        01EE    PARA    PUBLIC  'DATA'
STACK. . . . . . . . . . . . . .        0100    PARA    STACK   'STACK'

Symbols:

                N a m e                 Type    Value   Attr

ADDRESSA . . . . . . . . . . . .        L BYTE  0003    DATA
ADDRESSB . . . . . . . . . . . .        L BYTE  0004    DATA
ALT. . . . . . . . . . . . . . .        Number  0080
AREAAX . . . . . . . . . . . . .        L WORD  0007    DATA
AREACX . . . . . . . . . . . . .        L WORD  0009    DATA
ASCROLL. . . . . . . . . . . . .        L NEAR  01AB    CODE
BACKSP . . . . . . . . . . . . .        N PROC  01B4    CODE    Length =001C
BACKSPACE. . . . . . . . . . . .        Number  0008
BELL . . . . . . . . . . . . . .        Number  0007
BELL1. . . . . . . . . . . . . .        L NEAR  0029    CODE
BKSP . . . . . . . . . . . . . .        L NEAR  0032    CODE
BKSPB. . . . . . . . . . . . . .        L NEAR  00C4    CODE
BS_SUB . . . . . . . . . . . . .        N PROC  00B3    CODE    Length =001F
CLRTAB . . . . . . . . . . . . .        N PROC  031E    CODE    Length =0011
CLR_ROW2 . . . . . . . . . . . .        L NEAR  010C    CODE
CLR_SPA. . . . . . . . . . . . .        L NEAR  0111    CODE
CMD_PORTR. . . . . . . . . . . .        Number  01A2
CMD_PORTW. . . . . . . . . . . .        Number  01A0
CMPDOWN. . . . . . . . . . . . .        L NEAR  030F    CODE
COL_END. . . . . . . . . . . . .        L BYTE  000D    DATA
COL_VALUE. . . . . . . . . . . .        L BYTE  000C    DATA
COPYLCD. . . . . . . . . . . . .        N PROC  024F    CODE    Length =0023
COPYROW. . . . . . . . . . . . .        N PROC  0232    CODE    Length =001D
COUNT. . . . . . . . . . . . . .        L BYTE  0006    DATA
CRIGHT . . . . . . . . . . . . .        L NEAR  024C    CODE
CURRET . . . . . . . . . . . . .        L NEAR  0343    CODE
CUR_OFF. . . . . . . . . . . . .        L NEAR  0336    CODE
CUR_ON . . . . . . . . . . . . .        L NEAR  033E    CODE
CUR_ONOFF. . . . . . . . . . . .        N PROC  032F    CODE    Length =0017
DATA_PORTR . . . . . . . . . . .        Number  01A3
DATA_PORTW . . . . . . . . . . .        Number  01A1
DD . . . . . . . . . . . . . . .        L NEAR  005C    CODE
DISPLAY. . . . . . . . . . . . .        L NEAR  007A    CODE
DISPRIGHT. . . . . . . . . . . .        L NEAR  019A    CODE
DISPSCROLL . . . . . . . . . . .        L NEAR  0197    CODE
DISP_SUB . . . . . . . . . . . .        N PROC  017A    CODE    Length =0021
DOWNCODE . . . . . . . . . . . .        Number  00DA
DOWN_LCD . . . . . . . . . . . .        N PROC  029B    CODE    Length =0020
DRIGHT . . . . . . . . . . . . .        L NEAR  02B8    CODE
FF . . . . . . . . . . . . . . .        L NEAR  0012    CODE
FF_SUB . . . . . . . . . . . . .        N PROC  0088    CODE    Length =002B
FINE . . . . . . . . . . . . . .        L NEAR  01EB    CODE
FINISH . . . . . . . . . . . . .        L NEAR  01CA    CODE
```

```
FORMFEED . . . . . . . . . . . . .      Number  000C
IN_DATA. . . . . . . . . . . . . .      N PROC  01F3    CODE    Length =0006
IP_MEM . . . . . . . . . . . . . .      L WORD  036C    CODE
JNZ. . . . . . . . . . . . . . . .      L DWORD 0052    CODE
KEEP_CUR . . . . . . . . . . . . .      L BYTE  000B    DATA
LA . . . . . . . . . . . . . . . .      L NEAR  0070    CODE
LA_SUB . . . . . . . . . . . . . .      N PROC  0176    CODE    Length =0004
LCD_TABLE. . . . . . . . . . . . .      N PROC  021F    CODE    Length =0013
LEFTARROW. . . . . . . . . . . . .      Number  00D3
LF . . . . . . . . . . . . . . . .      L NEAR  003C    CODE
LF_ROW2. . . . . . . . . . . . . .      L NEAR  00DC    CODE
LF_SUB . . . . . . . . . . . . . .      N PROC  00D2    CODE    Length =0064
LINEFEED . . . . . . . . . . . . .      Number  000A
MOVCHAR. . . . . . . . . . . . . .      L NEAR  02F5    CODE
MOVLCD . . . . . . . . . . . . . .      N PROC  02D9    CODE    Length =002B
NEX_DATA . . . . . . . . . . . . .      L NEAR  00E6    CODE
NEX_ROW. . . . . . . . . . . . . .      L NEAR  016D    CODE
NOMAL. . . . . . . . . . . . . . .      L NEAR  02B5    CODE
NONCOPY. . . . . . . . . . . . . .      L NEAR  0292    CODE
NO_CHANG . . . . . . . . . . . . .      L NEAR  0132    CODE
OUTA . . . . . . . . . . . . . . .      L NEAR  01D4    CODE
OUT_FUN. . . . . . . . . . . . . .      N PROC  01D0    CODE    Length =001D
OUT_LCD. . . . . . . . . . . . . .      F PROC  0000    CODE    Length =0088
OUT_POSITION . . . . . . . . . . .      L NEAR  0126    CODE
OUT_VAL. . . . . . . . . . . . . .      N PROC  01ED    CODE    Length =0006
POP_R. . . . . . . . . . . . . . .      N PROC  0359    CODE    Length =0013
PUSH_R . . . . . . . . . . . . . .      N PROC  0346    CODE    Length =0013
RA . . . . . . . . . . . . . . . .      L NEAR  0066    CODE
RA_CTN . . . . . . . . . . . . . .      L NEAR  0161    CODE
RA_RET . . . . . . . . . . . . . .      L NEAR  0175    CODE
RA_ROW2. . . . . . . . . . . . . .      L NEAR  0167    CODE
RA_SUB . . . . . . . . . . . . . .      N PROC  014B    CODE    Length =002B
READ_DATA. . . . . . . . . . . . .      L NEAR  01EA    CODE
RES_LCD. . . . . . . . . . . . . .      N PROC  02BB    CODE    Length =001E
RES_UD . . . . . . . . . . . . . .      L NEAR  0319    CODE
RETURN . . . . . . . . . . . . . .      Number  000D
RIGHT. . . . . . . . . . . . . . .      L NEAR  007D    CODE
RIGHTARROW . . . . . . . . . . . .      Number  00C4
ROW. . . . . . . . . . . . . . . .      L WORD  0001    DATA
ROW00. . . . . . . . . . . . . . .      L BYTE  000E    DATA    Length =0014
ROW01. . . . . . . . . . . . . . .      L BYTE  0022    DATA    Length =0014
ROW02. . . . . . . . . . . . . . .      L BYTE  0036    DATA    Length =0014
ROW03. . . . . . . . . . . . . . .      L BYTE  004A    DATA    Length =0014
ROW04. . . . . . . . . . . . . . .      L BYTE  005E    DATA    Length =0014
ROW05. . . . . . . . . . . . . . .      L BYTE  0072    DATA    Length =0014
ROW06. . . . . . . . . . . . . . .      L BYTE  0086    DATA    Length =0014
ROW07. . . . . . . . . . . . . . .      L BYTE  009A    DATA    Length =0014
ROW08. . . . . . . . . . . . . . .      L BYTE  00AE    DATA    Length =0014
ROW09. . . . . . . . . . . . . . .      L BYTE  00C2    DATA    Length =0014
ROW10. . . . . . . . . . . . . . .      L BYTE  00D6    DATA    Length =0014
ROW11. . . . . . . . . . . . . . .      L BYTE  00EA    DATA    Length =0014
ROW12. . . . . . . . . . . . . . .      L BYTE  00FE    DATA    Length =0014
ROW13. . . . . . . . . . . . . . .      L BYTE  0112    DATA    Length =0014
ROW14. . . . . . . . . . . . . . .      L BYTE  0126    DATA    Length =0014
ROW15. . . . . . . . . . . . . . .      L BYTE  013A    DATA    Length =0014
```

```
ROW16. . . . . . . . . . . . . .        L BYTE  014E    DATA    Length =0014
ROW17. . . . . . . . . . . . . .        L BYTE  0162    DATA    Length =0014
ROW18. . . . . . . . . . . . . .        L BYTE  0176    DATA    Length =0014
ROW19. . . . . . . . . . . . . .        L BYTE  018A    DATA    Length =0014
ROW20. . . . . . . . . . . . . .        L BYTE  019E    DATA    Length =0014
ROW21. . . . . . . . . . . . . .        L BYTE  01B2    DATA    Length =0014
ROW22. . . . . . . . . . . . . .        L BYTE  01C6    DATA    Length =0014
ROW23. . . . . . . . . . . . . .        L BYTE  01DA    DATA    Length =0014
RT . . . . . . . . . . . . . . .        L NEAR  0046    CODE
RT_ROW1. . . . . . . . . . . . .        L NEAR  0144    CODE
RT_ROW2. . . . . . . . . . . . .        L NEAR  0142    CODE
RT_SUB . . . . . . . . . . . . .        N PROC  0136    CODE    Length =0015
R_DATA . . . . . . . . . . . . .        L BYTE  0005    DATA
SCRIGHT. . . . . . . . . . . . .        L NEAR  021E    CODE
SCROLL . . . . . . . . . . . . .        N PROC  019B    CODE    Length =0019
SCROLLER . . . . . . . . . . . .        N PROC  01F9    CODE    Length =0026
SCRRIGHT . . . . . . . . . . . .        L NEAR  01B3    CODE
SCR_CON. . . . . . . . . . . . .        L NEAR  0234    CODE
SRIGHT . . . . . . . . . . . . .        L NEAR  0230    CODE
SUBRIGHT . . . . . . . . . . . .        L NEAR  00D1    CODE
TABLP. . . . . . . . . . . . . .        L NEAR  0326    CODE
TEST_UD. . . . . . . . . . . . .        N PROC  0304    CODE    Length =001A
TUDRIGHT . . . . . . . . . . . .        L NEAR  031C    CODE
TWENTY . . . . . . . . . . . . .        L BYTE  0000    DATA
UDTEST . . . . . . . . . . . . .        L NEAR  001C    CODE
UPCODE . . . . . . . . . . . . .        Number  00C1
UP_LCD . . . . . . . . . . . . .        N PROC  0272    CODE    Length =0029
URIGHT . . . . . . . . . . . . .        L NEAR  0299    CODE
UU . . . . . . . . . . . . . . .        L NEAR  0050    CODE
WAIT . . . . . . . . . . . . . .        L NEAR  01D9    CODE

Warning Severe
Errors  Errors
0       0
```

```
Symbol Cross Reference              (# is definition)       Cref-1

ADDRESSA . . . . . . . . . . . . .   35#   277   285   289
ADDRESSB . . . . . . . . . . . . .   36#   278   279   290   292   293   299
ALT. . . . . . . . . . . . . . . .   20#    26    27    28    29
AREAAX . . . . . . . . . . . . . .   39#   166   225   408
AREACX . . . . . . . . . . . . . .   40#   167   226   595
ASCROLL. . . . . . . . . . . . . .  389    394#

BACKSP . . . . . . . . . . . . . .  258    262   405#  418
BACKSPACE. . . . . . . . . . . . .   25#   188
BELL . . . . . . . . . . . . . . .   21#   183
BELL1. . . . . . . . . . . . . . .  179    182#
BKSP . . . . . . . . . . . . . . .  184    187#
BKSPB. . . . . . . . . . . . . . .  257    260#
BS_SUB . . . . . . . . . . . . . .  190    252#  266   356

CLRTAB . . . . . . . . . . . . . .  246    659#  671
CLR_ROW2 . . . . . . . . . . . . .  275    281   292#
CLR_SPA. . . . . . . . . . . . . .  293#   300
CMD_PORTR. . . . . . . . . . . . .   15#   428
CMD_PORTW. . . . . . . . . . . . .   13#   424
CMPDOWN. . . . . . . . . . . . . .  641    644#
CODE . . . . . . . . . . . . . . .  158#   158   160   730
COL_END. . . . . . . . . . . . . .   43#   498   513   534   538
COL_VALUE. . . . . . . . . . . . .   42#   497   512   518   533   537
COPYLCD. . . . . . . . . . . . . .  530#   542   557
COPYROW. . . . . . . . . . . . . .  499    508#  525   535   539
COUNT. . . . . . . . . . . . . . .   38#   243   253   263   273   302   306   316   321   331   333   336   340   345
                                    366    367   373   391   395   416   555   598   613

CRIGHT . . . . . . . . . . . . . .  514    521#
CURRET . . . . . . . . . . . . . .  685    689#
CUR_OFF. . . . . . . . . . . . . .  682#
CUR_ON . . . . . . . . . . . . . .  681    686#
CUR_ONOFF. . . . . . . . . . . . .  168    550   596   677#  693

DATA . . . . . . . . . . . . . . .    9#     9   143   160   163   473   474
DATA_PORTR . . . . . . . . . . . .   16#   434   458
DATA_PORTW . . . . . . . . . . . .   14#   450
DD . . . . . . . . . . . . . . . .  207#
DISPLAY. . . . . . . . . . . . . .  219    222#
DISPRIGHT. . . . . . . . . . . . .  371    375   378#
DISPSCROLL . . . . . . . . . . . .  369    376#
DISP_SUB . . . . . . . . . . . . .  223    364#  380   626
DOWNCODE . . . . . . . . . . . . .   29#   208   645
DOWN_LCD . . . . . . . . . . . . .  210    569#  585   647
DRIGHT . . . . . . . . . . . . . .  573    578   581#

FF . . . . . . . . . . . . . . . .  172#
FF_SUB . . . . . . . . . . . . . .  175    233#  248
FINE . . . . . . . . . . . . . . .  437    440#
FINISH . . . . . . . . . . . . . .  410    415#
FORMFEED . . . . . . . . . . . . .   24#   173

IN_DATA. . . . . . . . . . . . . .  283    456#  460   516
IP_MEM . . . . . . . . . . . . . .  701    710   715   724   728#

JNZ. . . . . . . . . . . . . . . .  204#
```

```
 Symbol Cross Reference                 (# is definition)      Cref-2

KEEP_CUR . . . . . . . . . . . .   41#  556  597

LA . . . . . . . . . . . . . . .  214   217#
LA_SUB . . . . . . . . . . . . .  220   355#  358
LCD_TABLE. . . . . . . . . . . .  486   494#  503
LEFTARROW. . . . . . . . . . . .   27#  218   409
LF . . . . . . . . . . . . . . .  189   192#
LF_ROW2. . . . . . . . . . . . .  277#
LF_SUB . . . . . . . . . . . . .  195   272#  309   346   396
LINEFEED . . . . . . . . . . . .   22#  193

MOVCHAR. . . . . . . . . . . . .  621#  628
MOVLCD . . . . . . . . . . . . .  560   580   594   608#  633

NEX_DATA . . . . . . . . . . . .  279#  291
NEX_ROW. . . . . . . . . . . . .  332   344#
NOMAL. . . . . . . . . . . . . .  576   579#
NONCOPY. . . . . . . . . . . . .  554   558#
NO_CHANG . . . . . . . . . . . .  304   307#

OUTA . . . . . . . . . . . . . .  425#  451   459
OUT_FUN. . . . . . . . . . . . .  235   236   238   240   242   261   282   286   296   307   322   342   374   392
                                  407   414   422#  443   515   599   614   684   688

OUT_LCD. . . . . . . . . . . . .  159#  229   731
OUT_POSITION . . . . . . . . . .  295   301#
OUT_VAL. . . . . . . . . . . . .  288   298   365   412   448#  452

POP_R. . . . . . . . . . . . . .  227   714#  726
PUSH_R . . . . . . . . . . . . .  161   700#  712

RA . . . . . . . . . . . . . . .  209   212#
RA_CTN . . . . . . . . . . . . .  334   338#
RA_RET . . . . . . . . . . . . .  343   347#
RA_ROW2. . . . . . . . . . . . .  337   341#
RA_SUB . . . . . . . . . . . . .  215   330#  349
READ_DATA. . . . . . . . . . . .  435   438#
RES_LCD. . . . . . . . . . . . .  577   590#  603   650
RES_UD . . . . . . . . . . . . .  646   649#
RETURN . . . . . . . . . . . . .   23#  198
RIGHT. . . . . . . . . . . . . .  176   181   186   191   196   201   206   211   216   221   224#
RIGHTARROW . . . . . . . . . . .   26#  213
ROW. . . . . . . . . . . . . . .   34#  178   244   466   551   553   559   572   574   575   593   616
ROW00. . . . . . . . . . . . . .   47#  476   615   662
ROW01. . . . . . . . . . . . . .   51#  477
ROW02. . . . . . . . . . . . . .   55#
ROW03. . . . . . . . . . . . . .   59#
ROW04. . . . . . . . . . . . . .   63#
ROW05. . . . . . . . . . . . . .   67#
ROW06. . . . . . . . . . . . . .   71#
ROW07. . . . . . . . . . . . . .   75#
ROW08. . . . . . . . . . . . . .   79#
ROW09. . . . . . . . . . . . . .   83#
ROW10. . . . . . . . . . . . . .   87#
ROW11. . . . . . . . . . . . . .   91#
ROW12. . . . . . . . . . . . . .   95#
```

```
 Symbol Cross Reference              (# is definition)      Cref-3

ROW13. . . . . . . . . . . . . .   99#
ROW14. . . . . . . . . . . . . .  103#
ROW15. . . . . . . . . . . . . .  107#
ROW16. . . . . . . . . . . . . .  111#
ROW17. . . . . . . . . . . . . .  115#
ROW18. . . . . . . . . . . . . .  119#
ROW19. . . . . . . . . . . . . .  123#
ROW20. . . . . . . . . . . . . .  127#
ROW21. . . . . . . . . . . . . .  131#   496
ROW22. . . . . . . . . . . . . .  135#   532
ROW23. . . . . . . . . . . . . .  139#   536
RT . . . . . . . . . . . . . . .  194    197#
RT_ROW1. . . . . . . . . . . . .  319    321#
RT_ROW2. . . . . . . . . . . . .  317    320#
RT_SUB . . . . . . . . . . . . .  200    315#   324
R_DATA . . . . . . . . . . . . .   37#   284    287

SCRIGHT. . . . . . . . . . . . .  467    487#
SCROLL . . . . . . . . . . . . .  377    387#   399
SCROLLER . . . . . . . . . . . .  276    465#   489
SCRRIGHT . . . . . . . . . . . .  393    397#
SCR_CON. . . . . . . . . . . . .  511#   520
SRIGHT . . . . . . . . . . . . .  500#
STACK. . . . . . . . . . . . . .    2#     2      7
SUBRIGHT . . . . . . . . . . . .  255    259    264#

TABLP. . . . . . . . . . . . . .  664#   667
TEST_UD. . . . . . . . . . . . .  180    638#   654
TUDRIGHT . . . . . . . . . . . .  643    648    651#
TWENTY . . . . . . . . . . . . .   33#   245    617

UDTEST . . . . . . . . . . . . .  174    177#
UPCODE . . . . . . . . . . . . .   28#   203    640
UP_LCD . . . . . . . . . . . . .  205    547#   564    642
URIGHT . . . . . . . . . . . . .  552    561#
UU . . . . . . . . . . . . . . .  199    202#

WAIT . . . . . . . . . . . . . .  429#   431
```

The Assembler directive -- SEGMENT

In the beginning of this program, an assembler directive SEGMENT instructs the Macro Assembler to reserve a memory space of 256 bytes as the stack segment so that data can be saved in the stack segment before a CALL or JUMP instruction. You should refer to the Macro Assembler Manual for more details of the assembler directive SEGMENT.

Each time you use a SEGMENT directive to allocate a memory space to a segment, you have to use the directive ENDS to tell the assembler that it is the end of a segment.

The Data Segment

After the stack segment is set aside, another SEGMENT statement is used to define the data segment for the program. Data and variables to be processed in an 8088 assembly language program should be defined in the data segment.

The Assembler directive -- EQU

The EQU (EQUATE) directive assigns a value of an expression to a name. For example, the EQU statement

```
     CMD_PORTW        EQU  01A0H
```

assigns the hexadecimal value 1A0H to the name CMD_PORTW. In a source program, an H is affixed to a value to designate that the value is in hexadecimal.

The EQU directive sometimes takes the form of an equal sign =. As you can see from the example program, the first four EQU directives assign the I/O port addresses to the four LCD ports.

Values are assigned to control codes with the EQU directives.

The Assembler directive -- DEFINE

The DEFINE assembler directive assigns a pre-defined value to a byte or multiple of bytes according to the second letter of the directive. The DEFINE directives are represented by different mnemonics such as DB (DEFINE BYTE), DW (DEFINE WORD), DD (DEFINE DOUBLEWORD), DQ (DEFINE QUADWORD), and DT (DEFINE TENBYTES). You should refer to the Macro Assembler Manual for more details of that assembler directive.

Constants and variables are defined using the DEFINE directives. The LCD buffer are initialized to zeros with the DB directives.

The Code Segement

The code segement follows the ENDS directive for the data segment. Before the SEGMENT assembler directive, there is a comment field which defines the contents of some registers which should be set before entering the LCD routine. Here, we will explain the comment field in detail:

1) AL - holds the ASCII character (parameter) to be output to the LCD.

2) CH=Ø - indicates that you do not expect the cursor to appear on the LCD.

3) CH<>Ø - means the reverse of CH=Ø; i.e., you expect the cursor to appear on the display.

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

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

The example program consists of 23 subroutines, including the main program named OUT_LCD. The function of each subroutine is described as follows:

Name	Functions
1. OUT_LCD	The main program checking for the input data type.
2. FF_SUB	A subroutine processing the form feed code.
3. BS_SUB	A subroutine processing backspace initialization.
4. LF_SUB	A subroutine processing the line feed code.
5. RT_SUB	A subroutine processing the carriage return code.
6. RA_SUB	A subroutine processing the right-arrow code.
7. LA_SUB	A subroutine processing the left-arrow code.
8. DISP_SUB	A subroutine processing the cursor positioning after displaying a character.
9. SCROLL	A subroutine for decision-making on scrolling LCD when the end of the second row of the LCD screen is reached.
1Ø. BACKSP	A subroutine processing the backspace code.
11. OUT_FUN	A subroutine communicating with the LCD I/O ports.

Name	Functions
12. OUT_VAL	A subroutine for writing characters to I/O ports.
13. IN_DATA	A subroutine for reading characters from I/O ports.
14. SCROLLER	A subroutine for scrolling the contents of the buffer one line up.
15. LCD_TABLE	A subroutine for making some ·preparations for copying the images of the first row on the LCD to the 21st row of the buffer in memory.
16. COPYROW	A subroutine performing the copying operation.
17. COPY_LCD	A subroutine for making some preparations for copying the two rows of images on the LCD to the 22nd and 3rd rows of the buffer in memory respectively.
18. UP_LCD	A subroutine for scrolling the screen so that the upper part of the screen can be viewed.
19. DOWN_LCD	A subroutine for scrolling the screen so that the·lower part of the screen can be viewed.
2Ø. RES_LCD	A subroutine for restoring the original images back to the LCD screen from the 22nd and 23rd rows of the buffer.
21. MOVLCD	A subroutine for moving a line of characters in buffer to the LCD screen.
22. TEST_UP	A subroutine testing if the input parameter is ALT_A or ALT_Z.
23. CLRTAB	A subroutine clearing up all the contents of the buffer to blanks.

We can now look at the LCD program. After the comment field, a SEGMENT directive is used to set aside a memory space for the instruction code.

The first thing the program will do is to push the contents of all registers onto the stack. This is done by the instruction CALL PUSH_R. The procedure PUSH_R is listed in the example program as the last procedure.

Why must we push the contents of all registers onto the stack? Because we want to give the program flexibility so that it can be used with or called by other programs. We assume that this program can be called by another program.

One thing you have to keep in mind is that if your program is associated with (or called by) another program, you have to save onto the stack the current status (the results the calling program just produced before calling another program) of all registers before calling another program. Then as the called program finishes execution, the POP instruction is executed to restore the system status back to their original state. This practice ensures that when the called program finishes execution, program control will return to the calling program without destroying the status before calling.

An alternative to ensure that system status will be kept intact is to push the current status onto the stack as soon as a called program is executed. This is what the example program does to save the system status. You can adopt this programming technique in your own program.

In order to access the memory in the data segment (DS), the program initializes the DS register to point to the beginning of the DS (Data Segment) by the instruction MOV BX,DATA and MOV DS,BX.

Since the input parameters are important for subsequent operations, we use two word variables, namely AREAAX and AREACX, to save them in advance. The instruction CALL CUR_ONOFF determines whether to turn off the cursor based on the contents of the CH register input from the calling routine.

Initialize the LCD

Before sending a character to the LCD, you have to initialize the LCD by sending a set of values to the LCD. In our example program, the FF_SUB procedure can be used to initialize the LCD. The FF_SUB procedure outputs the following set of LCD initialization values -- 38H, ØDH, 6, and 1. The comments for the FF_SUB procedure explain briefly the function of these values. The LCD data sheet provides more information on why the LCD initialization values should be sent to the LCD.

The easiest way to initialize the LCD is using the following instructions:

```
     MOV  AL,ØCH
     CALL OUT_LCD    (OUT_LCD here is our example program.)
```

After initializing the LCD, then you can send a character to the LCD to display. For example, if you intend to display the character A, you can use the following instructions:

```
     MOV  AL,ØCH     ;Initialize the LCD.
     CALL OUT_LCD    ;(OUT_LCD here is our example program.)
     MOV  AL,41H     ;Load the AL register with the ASCII code  of
                      the character A.
     CALL OUT_LCD    ;(OUT_LCD here is our example program.)
```

Generally speaking the form feed code demands for the action of printer. But in our program, this code (ØCH) is to cause the system to reset the LCD and clear up the LCD screen. To the hardware of the LCD, resetting the LCD screen requires four actions - function set, display and cursor on/off set, mode set (cursor movement direction), and display clearance.

The value 38H in the first MOV instruction aims at setting the function of the LCD. The procedure OUT_FUN is called twice in the FF_SUB rountine in order to achieve the purpose of function setting.

Each time you want to output a character onto the LCD screen, you have to call the OUT_FUN routine which performs the actual output process. The procedure OUT_FUN is responsible for communicating with the I/O ports of the LCD.

The value ØDH is to set the LCD screen to be able to display images and to set the cursor to be able to blink after the system has been powered up.

The value 6 is to set the cursor to operate or scan from left to right. Last, the value 1 is used to clear up the LCD screen and set the cursor to the upper left-hand cornor of the LCD screen.

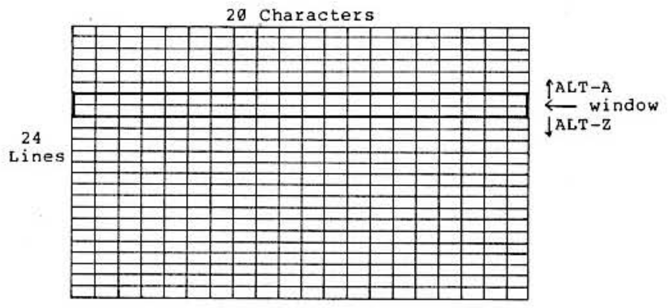
Now, the number of position where the cursor stays is 8ØH, representing the first column of the first row. Therefore, we move the value 8ØH into the variable COUNT which is used throughout all the associated LCD routines to indicate the current position of the cursor.

The subsequent instructions up to the end of the OUT_LCD routine check for the control characters to determine which routine should be executed. It should be easy for you to trace and understand these intructions.

Another important job FF_SUB performs is to move the constant 22 into the variable ROW. The variable ROW used in our program contains the current row number of the buffer in memroy whose contents are being shown on the LCD screen.

The Display Buffer

The buffer contains a total of 24 rows of lines (ranging from row Ø to row 23), and each line contains 2Ø columns. Thus, we get 48Ø bytes of memory, or 24*2Ø bytes. The display buffer can be visualized as follows:



The variable ROW is used as a pointer, which contains (always points to) the current row number of the display buffer whose contents are being shown on the LCD. The value of ROW is initialized to 22 after the FF_SUB routine is executed.

When you call the OUT_LCD procedure to output a character or characters to the LCD, the characters are stored beginning from row 22 of the display buffer (which corresponds to row 1 of the LCD). After both of the two rows of the LCD (which correspond to row 22 and row 23 of the display buffer) have been filled with characters, any further incoming characters to the LCD are displayed on the second row of the LCD, but the characters originally displayed on the first row of the LCD was shifted one line up into row 21 of the display buffer and the second row was shifted one line up into row 22 of the display buffer.

Each time the user enters the codes ALT_A or ALT_Z, the program will increment or decrement the variable ROW by one. In other words, the value of ROW will not change unless the codes ALT_A or ALT_Z are sent to the LCD.

The variable TWENTY represents the symbol of the value 2Ø which will be used in the associated routine (MOVLCD) to calcalate the starting address of a certain row of line in buffer required to output to the screen.

UDTEST will be executed only when the control codes ALT_A or ALT_Z are entered. Note that the CALL TEST_UD instruction in routine UDTEST will not be executed at the first calling of the LCD_OUT routine, because the contents of ROW was initialized to 22.

As you can see, the routines FF:, UDTEST:, and BELL1: through LA: all comes under the comment field "CONTROL CODE TEST". These routines test if a control code is entered. If the contents in the AL register does not match any of the control codes (such as bell, backspace, linefeed, return, and others supported by MPF-I/88), the program will fall through to the instruction labelled DISPLAY to output it onto the LCD screen.

Let us go on looking at the next routine called BS_SUB. The backspace control code is used to cause the cursor to move backward by one space on the same line. Two considerations in this routine should be taken into account. One is that when the cursor stays at the leftmost position of the first row, the backspace operation to the cursor must not occur. The other is that when the cursor stays at leftmost position of the second row, the cursor should skip to the rightmost position of the first row after the program recognizes the backspace code.

Now look at the LF_SUB routine. The linefeed control code that our program recognizes is used to cause the cursor on the LCD screen to advance by one line. If the cursor stays on the second line of the LCD screen, what we have to do is to move the current contents of the second line to the first line instead of causing the cursor to advance by one line. The routine labelled NEX_DATA performs the data movement operation. After making the cursor advance one line, we should clear the line (second line) which the cursor currently stays to blanks. Then the program will prompt the cursor at the position corresponding to the one where it stayed before.

Let us go on with the RT_SUB routine. The carriage return code is used to cause the cursor to stay at the beginning of the next line.

The RA_SUB routine advances the cursor by one space. The value 14 in the move instruction labelled RA_CTN is required by the hardware to advance the cursor to the right by one space. If the cursor stays at the rightmost position of the first line, the program will call the LF_SUB routine to move the cursor to the first column of the second row.

Next, look at the LA_SUB routine. This routine performs the same operation as the BS_SUB routine. However, when BS_SUB detects the backspace control code, it will clear the position preceding the current cursor position while moving the cursor. When the program encounters the leftarrow control code, it simply move the cursor backward by one space.

The DISP_SUB routine is used to output the character stored in the AL register. If the cursor reaches the end of the second row, the program will determine if the screen should scroll up based on the parameter in the CL register. Thus, the program calls the SCROLL routine to perform this job when the value in the AL register is equal to ØD4H.

It is considered not difficult for you to trace the SCROLL routine. Therefore, let us skip it over to the BACKSP routine. In this routine, we use a value 1ØH in the first move instruction which is required by hardware to move the cursor backward by one space. Then, the program will check which of these two control codes -- backspace and leftarrow, invokes this BACKSP routine. According to the logical judgement result, the program determines if it should perform a clean-up operation. There is another 1ØH value in the third move instruction of this routine; Its function is the same as the first move instruction. Because each time the program outputs a character onto the screen, the LCD hardware will automatically advance the cursor by one space. Thus, we have to rewrite the MOV AL,1ØH instruction after performing the clean-up instruction MOV AL,2ØH which is used to clear up the position where the cursor stayed last time..

Now, let us go to the OUT_FUN routine. This routine functions to interface with the four I/O ports and plays the actually output role in our program. This routine can be accessed from two entries- OUT_FUN and OUTA. Normally, this program is accessed from the entry OUT_FUN to output a character to the I/O ports. It can be also invoked by the IN_DATA routine to read in a character from I/O ports and invoked by the OUT_VAL routine to output a character to the I/O ports. The routine labelled WAIT is used to test if the LCD driver is busy at the time when we want to output or input a character to or from the LCD screen. If the LCD driver is busy, it returns a value in the AL register with the sign bit set to 1.

The SCROLLER routine is invoked when the screen is filled up with characters and the cursor cannot move down any more lines; i.e., once the cursor is on the bottom line (second), the screen should scroll up instead of moving the cursor down.

Before we replace the contents of the first line of the LCD screen with the contents of the second line, we have to move up the contents of the buffer in memory (from row 21 to row 1) by one row in order to move the contents of the first line of the LCD screen to row 21 of the buffer in memory. The SCROLLER, LCD_TABLE and COPYROW routines perform what we just stated.

The entire scrolling operation is accomplished with a string operation, using the MOVSB instruction.

The original contents of row Ø are always spoiled each time this routine is performed. Note that the use of the special assembler operator, **OFFSET**, in the MOV DI,OFFSET ROWØØ instruction. It provides us with the offset address of the variable ROWØØ.

The COPYLCD routine is invoked to move the contents of both the LCD screen lines to the row 22 and 23 of the display buffer in order to respond to the ALT_A or ALT_Z control code.

The UP_LCD routine is invoked by the ALT_A code. In this routine, the program uses the variable KEEP_CUR to record the cursor position the first time it receives the ALT_A control code in order to restore the cursor to its origianl position once the user enters any command or character except the ALT_A and ALT_Z control codes.

The DOWN_LCD routine does the reverse of the UP_LCD routine. However, the DOWN_LCD routine performs a decision-making process which is not performed by the UP_LCD routine. The decision-making process examines whether the ROW variable contains the value 22. If the ROW contains the value of 22, this means that the displaying of the LCD screen has already reached the buttom of the buffer in memory and no more down-scrolling can be performed.

The RES_LCD routine is used to move the orignal contents shown on the LCD screen from rows 22 and 23 of the buffer in memory back to the LCD screen and also restore the cursor to its origianl position based on the contents of the KEEP_CUR variable.

Let us keep going with the MOVLCD routine. This routine first calculates the starting address of the lines in the buffer to be output onto the LCD screen based on the value that the variable ROW contains, and then moves to the LCD screen two lines of contents (40 characters) in the buffer from the starting address it calculated.

The TEST_UD routine is used to determine which one of the ALT_A and ALT_Z codes is entered after one of them has been just entered once. If the code entered is not of one of them, the program will call the RES_LCD routine to restore the original images shown on the LCD screen.

Finally, let us see the CLRTAB routine. As its name implies, this routine is used to clear all the contents of the lines from row 0 to row 21 in the buffer to blanks.

Please take note that the above example program is assembled using Microsoft's Macro Assembler. Since the MPF-I/88 does not support Microsoft's Macro Assembler, the example program can not be entered and run on the MPF-I/88. However, you can adapt the example program to a form which can be run on the MPF-I/88. If you intend to do this, you have to change the lables and names into absolute addresses. Also, you are suggested to trace the OUT_LCD procedure contained in MPF-I/88 Monotor Program Source Listing, and compare that one with the example program.

5.5 Audio Interface Driver

The MPF-I/88 supports an audio interface circuit for buzzer output. Please refer to Sheet 2 of schematic diagram for the buzzer circuit. Bit 6 of port 18ØH is used to control the buzzer circuit. A sound is generated by applying a sequence of ones and zeros to this circuit.

You can visualize the buzzer as the paper cone of a speaker. To generate a sound, the paper cone must be attracted and released at high frequency by the audio interface circuit. To attract the paper cone, we apply a nominal voltage one (bit 1) to the audio interface circuit. To releae the paper cone, we apply a nominal voltage zero (bit Ø) to the audio interface circuit.

A nominal voltage one can be applied to the sound-generating circuit by using the OUT instruction to output a bit 1 to bit 6 of port 18ØH. A nominal voltage zero (bit Ø) can be applied to the audio interface circuit by outputting a bit Ø to bit 6 of port 18ØH.

You can locate the procedures BEEP and SOUND at lines 1552 and 1574 in the MPF-I/88 Monitor Program Source Listing. The subroutine which actually generate sound is labelled SOUND1:. As you can see from the comment field for the procedure SOUND, the BX can be loaded with a value that controls the frequency of the sound to be generated, while the CX register can be loaded with the value which controls the pitch of the sound to be generated.

As demonstrated in the MPF-I/88 Monitor Program Source Listing, you can use the SOUND procedure by including the INT 18H instruction in your own program. Before using the INT 18H statement, you can use the CX register to set the frequency of the sound we desire and the BX register to set the duration of the sound.

At the start of the BEEP subroutine, we move two initial values 2ØØH and 2ØH to the BX and CX registers, respectively. You can change them as you wish.

At this point, please refer to the chapter on I/O Programming of this manual for I/O port addresses where the function of bit 6 of the I/O port Ø18ØH is clearly described. Thereafter, you can understand why we set the constant label SPEAKER_IO to Ø18ØH and BEEP_BIT to 4ØH (=Ø1ØØØØØØ).

At the beginning of the program execution, we disable all the functional bits of port Ø18ØH so that the program execution might not be interrupted by outside devices, and at end of the execution we re-enable them. This point is very important to keep in mind when you write a program like this.

You can input a sound table using the DEFINE assembler directive. An example program is provided as follows. You can type in the example program and run it on your MPF-I/88.

This program when executed, will produce the basic music notes continuously. To stop the program, press the RESET key. The program will remain in the RAM after the RESET key was pressed.

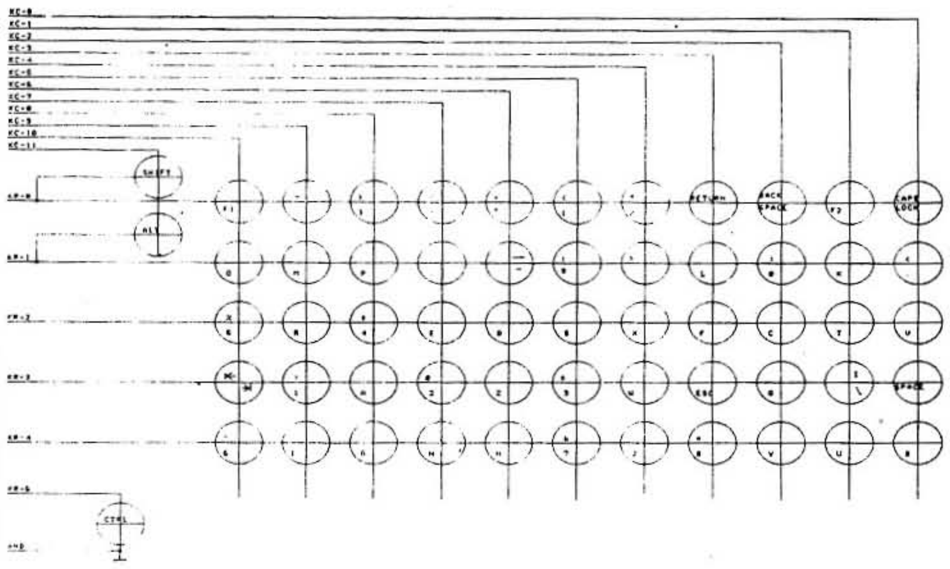
```
Address      Mnemonics    Operands     Comments
0080:0000    CALL         5            ;Invoke routine addressed by
                                        memory location 5.
0080:0003    JMP          0
0080:0005    MOV          SI,200       ;Move address 200 to SI
0080:0008    CLD
0080:0009    LODSB                     ;Move a byte of data addressed
                                        by the SI register into the AL
                                        register.
0080:000A    CMP          AL,1         ;Check if the end of the prede-
                                        fined data is encountered.
0080:000C    JNE          18           ;If data ends, jump to the
                                        instruction contained in
                                        memory location 18H.
0080:000E    LODSW                     ;Move a word of data addressed
                                        by the SI register into AX.
0080:000F    MOV          CX,AX        ;Move frequency into CX.
0080:0011    LODSW
0080:0012    MOV          BX,AX        ;Move music pitch into BX.
0080:0014    INT          18
0080:0016    JMP          9
0080:0018    RET

0080:0200    DB           1
0080:0201    DW           1D5,80
0080:0205    DB           1
0080:0206    DW           1B3,80
0080:020A    DB           1
0080:020B    DW           196,80
0080:020F    DB           1
0080:0210    DW           184,80
0080:0214    DB           1
0080:0215    DW           16B,80
0080:0219    DB           1
0080:021A    DW           155,80
0080:021E    DB           1
0080:021F    DW           148,80
0080:0223    DB           1
0080:0224    DW           136,80
0080:0228    DB           1
0080:0229    DW           114,80
0080:022D    DB           1
0080:022E    DW           F8,80
0080:0232    DB           1
0080:0233    DW           E6,80
0080:0237    DB           1
0080:0238    DW           B8,80
0080:023C    DB           1
0080:023D    DW           A2,80
0080:0241    DB           1
0080:0242    DW           9A,80
```

```
ØØ8Ø:Ø246  DB          1
ØØ8Ø:Ø247  DW          88,8Ø
ØØ8Ø:Ø24B  DB          1
ØØ8Ø:Ø24C  DW          78,8Ø
ØØ8Ø:Ø25Ø  DB          Ø
```

5.6 Keyboard Driver

A keyboard is an interface between the system and the outside world. Physically, the keyboard of the MPF-I/88 consists of 59 keys, including the space bar.

To understand the keyboard driver program, you need to refer to the schematic diagram for the keyboard, which shows the keyboard circuit. You will find that it resembles a matrix, consisting of 12 columns by five rows (12 x 5). Each node (intersection) of the column and row lines is assigned with one or two characters.



Each character supported by MPF-I/88 is assigned with a position code (scan code). The position code is a number between 1 and 71 with each uniquely identifing a specific key (there are 71 charaters supported by the MPF-I/88 the keyboard).

The keyboard driver program detects any change in the state of the keys by scanning (reading) the keyboard matrix every 15 ms.

Each time you enter a key from the keyboard, the keyboard program knows which key you are entering by examining the the position code (which is also generated by the keyboard program.) Tables 5-1 and 5-2 illustrate all of the 71 position codes with each corresponding ASCII code and character on the keyboard.

View of Table:

Input key
Position code
ASCII code

Table 5-1 Keyboard Position Code To ASCII Code
(Without holding down the SHIFT key)

PTA_TAB:

F1 (Ø) 81H	' (5) 6ØH	] (1Ø) 5DH	' (15) 27H	= (2Ø) 3DH	[(25) 5BH	/ (3Ø) 2FH	RET (35) ØDH	BKSP (4Ø) Ø8H	F2 (45) 82H	CAP (5Ø) 2ØH
O (1) 4FH	M (6) 4D	P (11) 5ØH	; (16) 3BH	— (21) 2DH	9 (26) 39H	. (31) 2EH	L (36) 4CH	Ø (41) 3ØH	K (46) 4BH	, (51) 2CH
5 (2) 35H	R (7) 52H	4 (12) 34H	E (17) 45H	D (22) 44H	S (27) 53H	X (32) 58H	F (37) 46H	C (42) 43H	T (47) 54H	V (52) 56H
--> (3) Ø9H	1 (8) 31H	A (13) 41H	2 (18) 32H	Z (23) 5AH	3 (28) 33H	W (33) 57H	ESC (38) 1BH	Q (43) 51H	\ (48) 5CH	SPACE (53) 2ØH
6 (4) 36H	I (9) 49H	G (14) 47H	N (19) 4EH	H (24) 48H	7 (29) 37H	J (34) 4AH	8 (39) 38H	Y (44) 59H	U (49) 55H	B (54) 42H

Table 5-2 Keyboard Position Code To ASCII Code
(With holding down the SHIFT key)

SHIFT:

F1 (Ø) 83H	~ (5) 7EH	} (1Ø) 7DH	" (15) 22H	+ (2Ø) 2BH	{ (25) 7BH	? (3Ø) 3FH	RET (35) ØDH	BKSP (4Ø) Ø8H	F2 (45) 84H	CAPS (5Ø) 2ØH
o (1) 6FH	m (6) 6DH	p (11) 7ØH	: (16) 3AH	- (21) 5FH	< (26) 28H	> (31) 3EH	l (36) 6CH	> (41) 29H	k (46) 6BH	< (51) 3CH
% (2) 25H	r (7) 72H	$ (12) 24H	e (17) 65H	d (22) 64H	s (27) 73H	x (32) 78H	f (37) 66H	c (42) 63H	t (47) 74H	v (52) 76H
<-- (3) Ø9H	! (8) 21H	a (13) 61H	@ (18) 4ØH	z (23) 7AH	# (28) 23H	w (33) 77H	ESC (38) 1BH	q (43) 71H	\ (48) 7CH	SPACE (53) 2ØH
^ (4) 5EH	i (9) 69H	g (14) 67H	n (19) 6EH	h (24) 68H	& (29) 26H	j (34) 6AH	* (39) 2AH	y (44) 79H	u (49) 75H	b (54) 62H

The keyboard program of MPF-I/88 is automatically invoked every 15 milliseconds by the CPU. The MPF-I/88 invokes the keyboard program in such a manner that every 15 milliseconds the timer chip 555 sends out a signal to interrupt the 8Ø88 processor through the NMI pin of the 8Ø88. Upon receipt of the interrupt signal, the 8Ø88 initiates the following events:

1) First, the 8Ø88 saves the machine status by pushing the contents of the Flags register onto the stack.

2) Next, the 8Ø88 clears the interrupt enable and trap bits in the Flags register to prevent subsequent maskable and single-step interrupts.

3) Then, the 8Ø88 establishes the interrupt routine return linkage by pushing the current CS and IP register contents onto the stack.

4) Finally, the 8Ø88 loads the CS and IP registers with the starting address of the keyboard program from the Interrupt Vector Table, and then accesses it.

It is the responsibility of the keyboard program to detect the keyboard interrupt and respond to it by returning a position code if a key is pressed.

On the MPF-I/88, the position code is generated by reading in a binary value which represents the key just being entered from the I/O port 1CØH. The position code itself may be interpreted in any manner desired. That is to say, the meaning of each key can be pre-defined by software.

Since the keyboard interrupt occurs asynchronously with respect to other program running in the computer, the striking of a key can occur at any time, and it is completely independent of when another program may wish to read keyboard. Our keyboard program is therefore required to save or buffer any keyboard input that it receives. To accomplish this, we use a "first-in, first-out" buffer, most often referred to as a "key queue".

A position code generated by the keyboard program is converted into a proper ASCII character code and then placed onto the key queue. When another program wishes to get keyboard input, it just takes the characters off the queue in the order in which they were received.

The size of the queue determines the maximum number of characters that can be buffered at any time. This represents the number of keystrokes you can type before causing the system to perform any operation.

Now we are going to explain how a keyboard scanning operation is performed. When reading the following paragraphs, please refer to the schematic diagram.

As with what we have stated before, there are 12 columns and five rows which result in a matrix on the keyboard circuit. Columns KC-Ø to KC-7 are physically assigned to I/O port Ø16ØH; and columns KC-8 to KC-11 are assigned to I/O port Ø18ØH. Next, let us see the row lines. Rows KR-Ø to KR-5 are assigned to I/O port Ø1CØH. Ports Ø16ØH and Ø18ØH are keyboard array outputs to the keyboard program; in reverse, they are inputs to the keyboard. Port Ø1CØH is a keyboard array output to the keyboard.

To find out if a key among all the keyboard keys is pressed, what we have to do is to start scanning from KC-11 through KC-Ø. A complete scanning operation from KC-11, KC-1Ø, KC-9 through KC-Ø is called a "scan-out" in our keyboard program.

In addition to column KC-11, each column of the keyboard matrix is scanned for five times. This is because during the scanning of each column, we have to scan five keys (from row KR-Ø to row KR-4 with the exception of column KC-11.) i.e., each column needs five scanning operations. At this point, you might ask how the keyboard program knows which column is required to scan at a certain time during scanning. Now, let us have a futher discussion about it; that is, indeed, only a programming technique.

Before the keyboard program starts scanning the keyboard, it will set column KC-11 to zero (low voltage) and the rest of columns to one (high voltage) by outputting to both the I/O ports Ø18ØH and Ø16ØH the value ØF7FFH (=1111Ø11111111111). In other words, the column which the program wishes to scan is pre-set to zero and the rest of columns to one, for the number of columns that the keyboard program can scan at a moment is only one column of five keys.

After scanning a column, the hardware (keyboard) will send out to port Ø1CØH a byte of value of which only one of the least significant five bits contains a zero value. Thus, the keyboard program can read that value into the AL register through the DX register which always connects to I/Ø ports. At this time, you can determine which key is pressed by shifting left the least significant five bits one by one to the Carry flag that we use as a "check-count" in our program. In our keyboard program, we also use a counter (namely, the DI register) to record the position of the key being pressed.

Through the value stored in the DI register, we can determine the position code of the key just being pressed which had been defined at the time when we designed the keyboard program. Then, we can also find the ASCII code of that key through a corresponding look-up ASCII code table defined in our program.

Our keyboard program is quite complicated and many factors should be taken into consideration, for it should normally handle many features, such as uppercase/lowercase characters, "ALT", "SHIFT" and "Shift-Lock" keys, and special control-key combinations. Thus, many tests and determinations are reqiured to make during the program execution.

There is also an important topic that should be stated here. That is the subject on the keybounce:

The keytops of the keyboard are usually depressed by hand. In general, the speed of the computer response to each of them is much faster than that of the human beings. No matter whether a key is pressed on the keyboard or not, the keyboard program must always scan the keyboard repeatedly. When being depressed or released, a key bounces for a short time. Fig. 5-1 is a time response diagram of typical key-depressing and key-releasing operation. Thus, a key-depression might be identified as two or more key-depressions if the keyboard scanning rate is too fast. To avoid this problem, the period of scanning we use in the program is longer than the bouncing time.

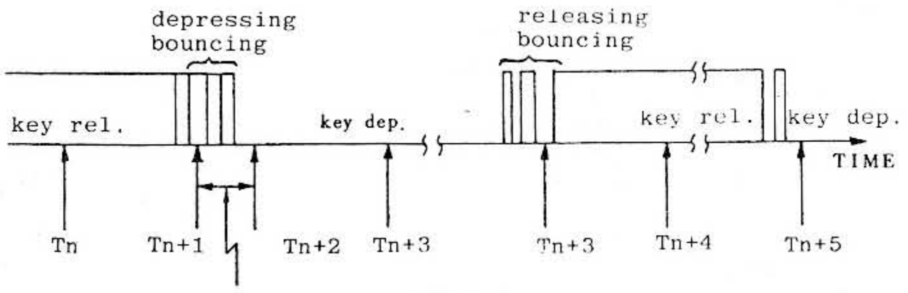


Fig 5-1

In Fig. 5-1, at the instant indicated by the upward arrow the key is examined. At Tn+2, the keyboard program found that the key was depressed and indentified the position code. At Tn+3, the key was also found depresseed. Since the key was found depressed in the previous scannings, the keyboard program will determine that this is not a new key-depression (i.e. the key has not been released during this time interval). Only if the key is found released at Tn+4 or Tn+5, a new key-depression will be really recognized at Tn+6.

A program for getting data from a keyboard designed by this rule will be immune from error, no matter how long the duration of the key-depression is and whatever is found at this period between Tn+1 and Tn+4 (Ø or 1).

In our keyboard program, we use also a variable as a repeat-count to test if a key is always depressed after the keyboard has been scanned out for 3Ø times. If yes, the same character will be shown on the screen. After that, if the program finds the same key still being depressed, it will output the same character onto the screen every 4 scan-out operations.

The MPF-I/88 keyboard interface program begins with the procedure KEY_NMI. You can locate it by referring to the cross reference section of the MPF-I/88 Monitor Program Source Listing. In order to let you understand keyboard interface programming more easily, an example program with a more detailed comment is provided as follows.

```
The Microsoft MACRO Assembler , Version 1.25               Page     1-1
                                                           12-18-84

                                        PAGE 60,132
0000                            DATA    SEGMENT PARA PUBLIC 'DATA'
                                ;
                                ;       I/O PORTS
                                ;
= 0180                          OPD_PORT1       EQU     0180H
= 0160                          OPD_PORT2       EQU     0160H
= 01C0                          IPD_PORT        EQU     01C0H
                                ;
                                ;       VARIABLES
                                ;
0000     0A [                   KEY_Q           DB      10      DUP(0)          ;ALLOCATE 10 BYTES OF MEMORY
               00
                  ]


                                                                                ;FOR THE KEY QUEUE BUFFER.
000A     0B [                   NEW_KEY_BUF     DB      11      DUP(0)          ;ALLOCATE 11 BYTES OF MEMORY FOR
               00
                  ]


                                                                                ;THE INPUTS JUST BEING KEYED IN.
0015     0A [                   OLD_KEY_BUF     DB      10      DUP(0)          ;ALLOCATE 10 BYTES OF MEMORY FOR
               00
                  ]


                                                                                ;THE INPUTS KEYED IN AT THE LAST
                                                                                ;SCAN-OUT OPERATION.
001F  0000                      NEW_NO_FLG      DW      0                       ;THE NUMBER OF INPUTS JUST BEING
                                                                                ;KEYED IN.
0021  0000                      OLD_NO_FLG      DW      0                       ;THE NUMBER OF INPUTS KEYED IN
                                                                                ;AT THE LAST SCAN-OUT OPERATION.
0023  00                        CAPS_COUNTER    DB      0                       ;CAPS LOCK COUNTER
0024  0000                      REP_COUNTER     DW      0                       ;FIRST REPEAT COUNTER
0026  0000                      REP_COUNTER1    DW      0                       ;REPEAT AGAIN COUNTER
0028  00                        LAST_KEY_FLG    DB      0                       ;A FLAG TO IDENTIFY IF ANY CHARACTER
                                                                                ;HAD BEEN TYPED IN AT THE LAST
                                                                                ;SCANNNG OPERATION. IF YES, A VALUE
                                                                                ;OF "FF" IS MOVED INTO THIS VARIABLE.
0029  00                        CTRL_P_COUNTER  DB      0                       ;CTRL-P COUNTER
002A  00                        NO_KEY_COUNTER  DB      0                       ;A COUNTER FOR THE RECORD OF HOW
                                                                                ;MANY TIMES OF SCAN-OUT OPERATION
                                                                                ;A CHARACTER HAS BEEN NOT DETECTED.
002B  00                        SPECIAL         DB      0                       ;CTRL,SHIFT,ALT FLAG
002C  00                        PTR_FLG         DB      0                       ;PRINTER FLAG
002D  00                        CAPS_LOCK       DB      0                       ;CAPS LOCK FLAG
002E                            DATA    ENDS
                                ;
0000                            CODE    SEGMENT PARA PUBLIC 'CODE'
0000                            KEY_NMI PROC    FAR
                                        ASSUME  CS:CODE,DS:DATA,ES:DATA
0000  1E                                PUSH    DS
0001  33 C0                             XOR     AX,AX                           ;EQUIVALENT OF "MOV AX,0"
0003  50                                PUSH    AX
                                ;
                                ; The above three instructions are used to store onto the stack the address
                                ; of a instruction next to the one that the MS-DOS is executing when we start
                                ; to run our keyboard program. With the address stored in the STACK segment,
                                ; the system can return back to the MS-DOS prompt once the program ends with
                                ; its last instruction "IRET".

ØØØ4  8E D8                          MOV     DS,AX                       ;SET A VECTOR FOR THE NMI_IN ROUTINE
                                                                         ;IN THE VECTOR TABLE WHICH STARTS FROM
                                                                         ;BEGINNING (Ø) OF THE DATA SEGMENT (DS).
ØØØ6  B8 ---- R                      MOV     AX,SEG NMI_IN
ØØØ9  A3 ØØØA                        MOV     DS:[ØAH],AX
ØØØC  B8 ØØ24 R                      MOV     AX,OFFSET NMI_IN
ØØØF  A3 ØØØ8                        MOV     DS:[8],AX
ØØ12  B8 ---- R                      MOV     AX,DATA
ØØ15  8E D8                          MOV     DS,AX                       ;THE GET_KEY ROUTINE WILL USE
                                                                         ;THE DS REGISTER, SO THE DS IS SET HERE
                                                                         ;IN ADVANCE.

ØØ17  BA Ø18Ø                        MOV     DX,OPD_PORT1                ;ENABLE THE NMI OF THE 8Ø88 BY
                                                                         ;SETTING THE FIFTH BIT (BIT 4) OF
                                                                         ;I/O PORT 18ØH TO ONE.
ØØ1A  BØ FF                          MOV     AL,ØFFH
ØØ1C  EE                             OUT     DX,AL
ØØ1D                            AGAIN:
ØØ1D  E8 Ø2E5 R                      CALL    GET_KEY                     ;FETCH A CHARACTER FROM THE KEY QUEUE
                                                                         ;BUFFER.
ØØ2Ø  CD Ø9                          INT     9H                          ;TO PLACE IT ON THE SCREEN.
ØØ22  EB F9                          JMP     AGAIN
ØØ24                            NMI_IN:
ØØ24  E8 Ø22D R                      CALL    PUSH_R                      ;PUSH ALL REGISTERS ONTO THE STACK.
ØØ27  B8 ---- R                      MOV     AX,DATA
ØØ2A  5Ø                             PUSH    AX
ØØ2B  5Ø                             PUSH    AX
ØØ2C  1F                             POP     DS
ØØ2D  Ø7                             POP     ES
                                ;
                                ;THE KEYBOARD SCANNING OPERATION STARTS HERE, THREE OUTPUT PORTS ARE USED IN THE "SCAN" ROU
                     TINE.
                                ;THEY ARE: OUTPUT PORTS Ø18ØH AND Ø16ØH, INTPUT PORT Ø1CØH
                                ;
ØØ2E                            SCAN:
ØØ2E  B8 F7FF                        MOV     AX,ØF7FFH                   ;OUT AH TO PORT Ø18ØH; AL TO PORT Ø16ØH
ØØ31  BA Ø16Ø                        MOV     DX,OPD_PORT2                ;DX POINTS TO PORT Ø16ØH
ØØ34  EE                             OUT     DX,AL
ØØ35  86 EØ                          XCHG    AH,AL                       ;EXCHANGE AH AND AL
ØØ37  BA Ø18Ø                        MOV     DX,OPD_PORT1                ;DX POINTS AT PORT Ø18ØH
ØØ3A  EE                             OUT     DX,AL
ØØ3B  BA Ø1CØ                        MOV     DX,IPD_PORT
ØØ3E  EC                             IN      AL,DX                       ;OBTAIN AN INPUT VALUE FROM PORT Ø1CØH AND
                                                                         ;PASS IT ONTO THE AL REGISTER.E
ØØ3F  C6 Ø6 ØØ2B R ØØ                MOV     SPECIAL,Ø                   ;CLEAR SPECIAL WHICH WILL
                                                                         ;BE USED TO SAVE THE FLAG
                                                                         ;OF CTRL, SHIFT AND ALT
ØØ44  A8 2Ø                          TEST    AL,2ØH                      ;TEST IF CTRL KEY IS PRESSED.
ØØ46  75 Ø5                          JNZ     CHK_SHIFT
ØØ48  8Ø ØE ØØ2B R Ø4                OR      SPECIAL,4                   ;CTRL KEY PRESSED, SAVE SPECIAL WITH THE VA
```

```
                                   LUE 4H

       004D                              CHK_SHIFT:
       004D  D0 D8                              RCR     AL,1                          ;CHECK IF SHIFT KEY PRESSED?
       004F  72 08                              JC      CHK_ALT1
       0051  80 0E 002B R 01                    OR      SPECIAL,1                     ;SET SPECIAL'S BIT FOR SHIFT, ALT, AND CTRL
                                    KEYS
       0056  EB 0A 90                           JMP     KCOL                          ;SPECIAL=1, MEANING THAT THE SHIFT KEY
                                                                                      ;HAS BEEN PRESSED.

       0059                              CHK_ALT1:                                    ;SPECIAL=2, MEANING THAT ALT PRESSED
       0059  D0 D8                              RCR     AL,1                          ;SPECIAL=4, MEANING THAT CTRL PRESSED
       005B  72 05                              JC      KCOL
       005D  80 0E 002B R 02                    OR      SPECIAL,2
       0062                              KCOL:
       0062  B8 FBFF                            MOV     AX,0FBFFH                     ;PREPARE THE AX WITH THE VALUE FBFFH FOR
                                                                                      ;OUTPUT PORT.
       0065  BF 0000                            MOV     DI,0                          ;DI REPRESENTS THE POSITION CODE COUNTER.
       0068                              KCOL1:
       0068  BA 0160                            MOV     DX,OPD_PORT2
       006B  EE                                 OUT     DX,AL                         ;OUTPUT THE AL ONTO PORT 0160H
       006C  50                                 PUSH    AX
       006D  86 E0                              XCHG    AH,AL
       006F  BA 0180                            MOV     DX,OPD_PORT1                  ;OUTPUT THE AH ONTO PORT 0180H
       0072  EE                                 OUT     DX,AL
       0073  BA 01C0                            MOV     DX,IPD_PORT
       0076  EC                                 IN      AL,DX
       0077  B1 05                              MOV     CL,5                          ;CL IDENTIFIES THE ROW NUMBER OF KEYBOARD
                                                                                      ;MATRIX TO BE SCANNED.

       0079  D0 F8                       KROW:  SAR     AL,1                          ;SHIFT THE AL ONE BIT TO THE RIGHT
       007B  73 0E                              JNC     SHORT KEY_DN                  ;NO CARRY MEANS THAT A KEY JUST ENTERED
                                                                                      ;HAS BEEN DETECTED.

       007D                              FIND_NEXT_KEY:
       007D  47                                 INC     DI                            ;INCREASE THE POSITION COUNTER BY ONE.
       007E  80 E9 01                           SUB     CL,1                          ;DECREASE THE ROW NUMBER BY ONE IN
                                                                                      ;ORDER TO SCAN THE NEXT ROW.
       0081  75 F6                              JNZ     SHORT KROW                    ;IS THE SCANNING OF ALL OF THE 5 ROWS
                                                                                      ;FINISHED?
       0083  58                                 POP     AX
       0084  D1 F8                              SAR     AX,1
       0086  72 E0                              JC      SHORT KCOL1                   ;IS THE SCANNING OF ALL THE COLUMNS
                                                                                      ;FINISHED?
       0088  E9 01B4 R                          JMP     SCAN_OUT
       008B                              KEY_DN:
       008B  E8 022D R                          CALL    PUSH_R
       008E  83 FF 32                           CMP     DI,50                         ;IS CAPS LOCK KEY PRESSED?
       0091  75 18                              JNE     KEY_DN_1                      ;GO ON SEARCHING FOR THE NEXT CODE
       0093  80 3E 0023 R 00                    CMP     CAPS_COUNTER,0                ;IS IT THE FIRST TIME FOR THE "CAPS LOCK"
                                                                                      ;KEY TO ENTER?
       0098  74 05                              JE      CAL_CAPS
       009A  E8 0240 R                          CALL    POP_R
       009D  EB DE                              JMP     FIND_NEXT_KEY
```

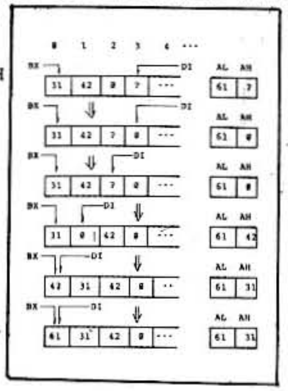
```
ØØ9F                          CAL_CAPS:
ØØ9F  FE Ø6 ØØ23 R                    INC     CAPS_COUNTER                ;CAUSE THE CAPS LOCK NOT TO BE ABLE
                                                                          ;TO REPEAT.
ØØA3  E8 Ø25F R                       CALL    CAPS
ØØA6  E8 Ø24Ø R                       CALL    POP_R
ØØA9  EB D2                           JMP     FIND_NEXT_KEY

ØØAB                          KEY_DN_1:
ØØAB  F6 Ø6 ØØ2B R Ø1                 TEST    SPECIAL,1                   ;TEST IF SHIFT KEY IS PRESSED BY
                                                                          ;BY USING LOGIC "AND" WITHOUT
                                                                          ;DESTORYING THE CONTENTS OF SPECIAL.

ØØBØ  74 17                           JZ      NOSHIFT                     ;ZERO FLAG SET TO Ø MEANS THAT SPECIAL
                                                                          ;DOESN'T CONTAIN THE VALUE OF 1.

ØØB2  F6 Ø6 ØØ2B R Ø4                 TEST    SPECIAL,4H                  ;SHIFT KEY PRESSED, CHECK AGAIN IF
                                                                          ;CTRL KEY IS PRESSED
ØØB7  74 Ø8                           JZ      NOCTRL
ØØB9  BB Ø2FC R                       MOV     BX,OFFSET PTA_TAB           ;BOTH OF SHIFT AND CTRL KEY ARE PRESSED
                              ;
                              ;(OPERATOR "OFFSET" PROVIDES US WITH THE OFFSET ADDRESS OF THE VARIABLE PTA_TAB.)
                              ;
ØØBC  2E: 8A Ø1                       MOV     AL,CS: [BX+DI]              ;MOVE THE CORRESPONDING ASCII CODE
                                                                          ;OF THE "PTA_TAB" TABLE INTO THE AL.
ØØBF  EB 15                           JMP     SHORT CHECK_CTRL_P
ØØC1                          NOCTRL:                                     ;NO CTRL KEY, SFIFT KEY ONLY
ØØC1  BB Ø333 R                       MOV     BX,OFFSET SHIFT             ;MOVE THE CORRESPONDING ASCII CODE
                                                                          ;OF THE "SHIFT" TABLE INTO THE AL.
ØØC4  2E: 8A Ø1                       MOV     AL,CS: [BX+DI]
ØØC7  EB 2E                           JMP     SHORT CHK_ALT
ØØC9                          NOSHIFT:                                    ;NO SHIFT KEY PRESSED, CHECK CTRL KEY
ØØC9  BB Ø2FC R                       MOV     BX,OFFSET PTA_TAB
ØØCC  2E: 8A Ø1                       MOV     AL,CS: [BX+DI]              ;NO SHIFT KEY TAKE ASCII INTO AL
ØØCF  F6 Ø6 ØØ2B R Ø4                 TEST    SPECIAL,4H                  ;TEST CTRL KEY
ØØD4  74 21                           JZ      CHK_ALT

ØØD6                          CHECK_CTRL_P:
ØØD6  3C 5Ø                           CMP     AL,'P'
ØØD8  75 1B                           JNE     AND_9FH
ØØDA  8Ø 3E ØØ29 R ØØ                 CMP     CTRL_P_COUNTER,Ø            ;IS IT THE FIRST TIME FOR THE
                                                                          ;"CTRL_P" CODE TO ENTER?
ØØDF  74 Ø5                           JE      CTRL_P_1
ØØE1  E8 Ø24Ø R                       CALL    POP_R
ØØE4  EB 97                           JMP     FIND_NEXT_KEY
ØØE6                          CTRL_P_1:
ØØE6  FE Ø6 ØØ29 R                    INC     CTRL_P_COUNTER              ;CAUSE THE CTRL_P NOT TO BE ABLE
                                                                          ;TO REPEAT TWICE.
ØØEA  BØ FF                           MOV     AL,ØFFH
ØØEC  3Ø Ø6 ØØ2C R                    XOR     PTR_FLG,AL                  ;SET CTRL P FLAG
ØØFØ  E8 Ø24Ø R                       CALL    POP_R
ØØF3  EB 88                           JMP     FIND_NEXT_KEY
                              ;
                              ;AFTER CONFIRMING THAT A "CRTL" KEY HAS BEEN PRESSED, PERFORM THE LOGICAL "AND"
                              ;INSTRUCTION TO OBTAIN THE ASCII CODE OF A CERTAIN CONTROL CHARACTER.
                              ;
```

```
ØØF5                      AND_9FH:
ØØF5   24 9F                      AND     AL,9FH
                          ;
ØØF7                      CHK_ALT:
ØØF7   F6 Ø6 ØØ2B R Ø2            TEST    SPECIAL,2H                      ;CHECK IF THE "ALT" IS PRESSED.
ØØFC   74 Ø6                      JZ      CHK_CAPS
ØØFE   3C 8Ø                      CMP     AL,8ØH                          ;THE ALT ASCII CODE = 8ØH
Ø1ØØ   76 Ø2                      JBE     CHK_CAPS
Ø1Ø2   Ø4 8Ø                      ADD     AL,8ØH                          ;ALT PRESSED;
                          ;
                          ;THE CORRESPONDING ASCII CODE FOR THE FUNCTIONAL (CONTROL) CHARACTER IS THE SUM
                          ;OF 8ØH PLUS THE ASCII CODE OF THE CHARACTER JUST BEING ENTERED.

Ø1Ø4                      CHK_CAPS:
Ø1Ø4   8Ø 3E ØØ2D R ØØ            CMP     CAPS_LOCK,Ø                     ;CHECK CAPS_LOCK FLG
Ø1Ø9   74 1E                      JE      CHK_OLD_NO
Ø1ØB   F6 Ø6 ØØ2B R Ø1            TEST    SPECIAL,1                       ;CHECK IF THE "SHIFT" KEY IS PRESSED.
Ø11Ø   75 ØD                      JNZ     SHIFT_CAPS_LOCK
Ø112   3C 41                      CMP     AL,41H                          ;CAPS_LOCK ENTERED ONLY, NO SHIFT
Ø114   72 13                      JB      CHK_OLD_NO                      ;CHECK IF ANY OF THE CAPITAL LETTERS
                                                                          ;(A - Z) IS PRESSED.
Ø116   3C 5A                      CMP     AL,5AH
Ø118   77 ØF                      JA      CHK_OLD_NO
Ø11A   Ø4 2Ø                      ADD     AL,2ØH                          ;TO GENERATE THE ASCII CODE OF A
                                                                          ;LOWERCASE LETTER.
Ø11C   EB ØB 9Ø                   JMP     CHK_OLD_NO

Ø11F                      SHIFT_CAPS_LOCK:
Ø11F   3C 61                      CMP     AL,61H                          ;CHECK IF ANY OF THE LOWERCASE LETTERS
Ø121   72 Ø6                      JB      CHK_OLD_NO                      ;IS PRESSED.
Ø123   3C 7A                      CMP     AL,7AH
Ø125   77 Ø2                      JA      CHK_OLD_NO
Ø127   2C 2Ø                      SUB     AL,2ØH                          ;TO OBTAIN THE ASCII CODE OF
                                                                          ;A CAPITAL LETTER.
Ø129                      CHK_OLD_NO:
Ø129   FE Ø6 ØØ1F R               INC     BYTE PTR NEW_NO_FLG             ;TO RECORD HOW MANY NEW KEYS HAS BEEN
                                                                          ;ENTERED WITHIN A SCAN-OUT OPERATION.
Ø12D   8Ø 3E ØØ21 R ØØ            CMP     BYTE PTR OLD_NO_FLG,Ø           ;CHECK IF ANY KEY HAD BEEN ENTERED AT
                                                                          ;THE LAST SCAN-OUT OPERATION.
Ø132   74 54                      JE      FILL_IN_NEW_BUF_AND_Q           ;IF YES, NO MORE CHECKS ARE REQUIRED,
                                                                          ;JUST MOVE THE ASCII CODE OF THE KEY
                                                                          ;JUST NOW ENTERED INTO "NEW_KEY_BUF"
                                                                          ;AND "KEY_Q".
Ø134   E8 Ø253 R                  CALL    COMP                            ;THERE IS A KEY PRESSED BEFORE,
                                                                          ;CHECK IF IT IS A NEW KEY.
Ø137   75 4F                      JNZ     FILL_IN_NEW_BUF_AND_Q           ;SAME AS OLD KEY ?
Ø139   8Ø 3E ØØ28 R FF            CMP     LAST_KEY_FLG,ØFFH               ;CHECK IF THE LAST KEY HAS BEEN PRESSED.
Ø13E   74 34                      JE      FILL_1
Ø14Ø   3A Ø6 ØØ15 R               CMP     AL,BYTE PTR [OLD_KEY_BUF]       ;NEW KEY SAME AS OLD KEY,CHECK
Ø144   75 Ø8                      JNE     FILL_IN_NEW_BUF                 ;IS IT LAST KEY ?
Ø146   C6 Ø6 ØØ28 R FF            MOV     LAST_KEY_FLG,ØFFH               ;FFH MEANS THAT THERE IS A KEY PRESSED
Ø14B   EB Ø8 9Ø                   JMP     FILL_Ø                          ;AT THE LAST SCANNING, AND VICE VERSA.


Ø14E                      FILL_IN_NEW_BUF:
```

```
Ø14E  8Ø 3E ØØ28 R FF                CMP   LAST_KEY_FLG,ØFFH          ;CHECK IF THE NEW KEY HAS BEEN PRESSED.
Ø153  74 1F                          JE    FILL_1
Ø155                          FILL_Ø:
Ø155  8D 3E ØØØA R                   LEA   DI,NEW_KEY_BUF             ;LAST KEY NOT PRESSED BEFORE
Ø159  8B DF                          MOV   BX,DI
Ø15B  E8 Ø266 R                      CALL  CHK_BUF
Ø15E  72 4D                          JC    BUF_OR_Q_FULL
Ø16Ø  E8 Ø278 R                      CALL  MV_DATA
Ø163  8Ø 3E ØØ28 R FF                CMP   LAST_KEY_FLG,ØFFH          ;IS IT THE LAST KEY?
Ø168  75 18                          JNE   CHK_OLD_NO_1
Ø16A                          INC_1:
Ø16A  F8                             CLC
Ø16B  FF Ø6 ØØ24 R                   INC   REP_COUNTER                ;YES, IT IS THE LAST KEY. INCREASE
                                                                      ;THE REP_COUNTER BY ONE.
Ø16F  72 F9                          JC    INC_1                      ;REP_COUNTER OVERFLOWS?
Ø171  EB ØF 9Ø                       JMP   CHK_OLD_NO_1
Ø174                          FILL_1:
Ø174  8D 3E ØØØB R                   LEA   DI,NEW_KEY_BUF+1           ;LAST KEY PRESSED BEFORE
Ø178  8B DF                          MOV   BX,DI                      ;PLACE THE OLD KEY AT THE 2ND POSITION
Ø17A  E8 Ø266 R                      CALL  CHK_BUF                    ;OF THE NEW_KEY_BUF.
Ø17D  72 2E                          JC    BUF_OR_Q_FULL
Ø17F  E8 Ø278 R                      CALL  MV_DATA

Ø182                          CHK_OLD_NO_1:
Ø182  E8 Ø24Ø R                      CALL  POP_R                      ;IT IS NOT THE LAST KEY, GO ON SEARCHING
Ø185  E9 ØØ7D R                      JMP   FIND_NEXT_KEY              ;FOR THE NEXT KEY.

Ø188                          FILL_IN_NEW_BUF_AND_Q:
Ø188  C6 Ø6 ØØ28 R FF                MOV   LAST_KEY_FLG,ØFFH
Ø18D  BF ØØØA R                      MOV   DI,OFFSET NEW_KEY_BUF
Ø19Ø  8B DF                          MOV   BX,DI
Ø192  E8 Ø266 R                      CALL  CHK_BUF                    ;CHECK IF NEW_KEY_BUF IS FULL.
Ø195  72 16                          JC    BUF_OR_Q_FULL
Ø197  E8 Ø278 R                      CALL  MV_DATA                    ;NEW_KEY_BUF NOT FULL YET, MOVE THE ASCII
                                                                      ;CODE OF THE KEY JUST ENTERED INTO IT.
Ø19A  BF ØØØØ R                      MOV   DI,OFFSET KEY_Q            ;MOVE THE AL TO THE KEY_Q.
Ø19D  8B DF                          MOV   BX,DI
Ø19F  E8 Ø266 R                      CALL  CHK_BUF                    ;CHECK IF THE KEY_Q IS FULL.
Ø1A2  72 Ø9                          JC    BUF_OR_Q_FULL
Ø1A4  E8 Ø278 R                      CALL  MV_DATA                    ;KEY_Q NOT FULL YET, MOVE THE ASCII CODE
                                                                      ;OF THE KEY JUST ENTERED INTO IT.
Ø1A7  E8 Ø24Ø R                      CALL  POP_R
Ø1AA  E9 ØØ7D R                      JMP   FIND_NEXT_KEY

Ø1AD                          BUF_OR_Q_FULL:
Ø1AD  E8 Ø24Ø R                      CALL  POP_R                      ;BUFFER OR QUEUE IS FULL, REJECT TO
Ø1BØ  58                             POP   AX                         ;ACCEPT ANY KEYS FROM THE KEYBOARD.
Ø1B1  EB 75 9Ø                       JMP   REN_NMI

Ø1B4                          SCAN_OUT:
Ø1B4  C6 Ø6 ØØ28 R ØØ                MOV   LAST_KEY_FLG,Ø
Ø1B9  8Ø 3E ØØ1F R ØØ                CMP   BYTE PTR NEW_NO_FLG,Ø      ;CHECK IF ANY KEY ENTERED.
Ø1BE  74 65                          JE    CAL_NO_KEY                 ;IF NOT, JUMP TO ROUTINE CAL_NO_KEY.
Ø1CØ  C6 Ø6 ØØ2A R ØØ                MOV   NO_KEY_COUNTER,Ø
Ø1C5  83 3E ØØ21 R ØØ                CMP   OLD_NO_FLG,Ø
```

```
01CA  74 27                        JE      SCAN1
01CC  A0 000A R                    MOV     AL,BYTE PTR [NEW_KEY_BUF]        ;CHECK IF THE LAST KEY IN THE NEW_KEYBUF IS

01CF  3A 06 0015 R                 CMP     AL,BYTE PTR [OLD_KEY_BUF]        ;THE SAME AS THE ONE IN THE OLD_KEY_BUF. TH
                      E
                                                                            ;LAST KEY IS ALWAYS PLACED AT THE FIRST
                                                                            ;POSITION OF THE CORRESPONDING BUFFER.

01D3  74 24                        JE      CHK_REP_COUNTER                  ;LAST KEY SAME AS BEFORE, JUMP TO ROUTINE
                                                                            ;CHK_REP_COUNTER TO CHECK FOR THE DELAY TIM
                      E.
01D5  8B 1E 001F R                 MOV     BX,NEW_NO_FLG
01D9  3B 1E 0021 R                 CMP     BX,OLD_NO_FLG
01DD  73 0E                        JAE     SCAN0
01DF  8D 3E 0000 R                 LEA     DI,KEY_Q
01E3  8B DF                        MOV     BX,DI
01E5  E8 0266 R                    CALL    CHK_BUF
01E8  72 C3                        JC      BUF_OR_Q_FULL
01EA  E8 0278 R                    CALL    MV_DATA
01ED                       SCAN0:
01ED  C7 06 0024 R 0000            MOV     REP_COUNTER,0
01F3                       SCAN1:
01F3  E8 02C5 R                    CALL    TRANSFER                         ;TRANSFER NEW_KEY_BUF TO OLD_KEY_BUF
01F6  EB 30 90                     JMP     REN_NMI

01F9                       CHK_REP_COUNTER:
01F9  83 3E 0024 R 1E              CMP     REP_COUNTER,30                   ;IS REP_COUNTER LARGER THAN 30?
01FE  72 F3                        JB      SCAN1                            ;REP_COUNTER < 30
0200  74 11                        JE      FIRST_REP
0202  FF 06 0026 R                 INC     REP_COUNTER1                     ;INCREASE SECOND REPEATEED COUNTER
0206  83 3E 0026 R 04              CMP     REP_COUNTER1,4                   ;SECOND REPEATED COUNTER > 4
020B  72 E6                        JB      SCAN1
020D  C7 06 0026 R 0000            MOV     REP_COUNTER1,0

0213                       FIRST_REP:
0213  BF 0000 R                    MOV     DI,OFFSET KEY_Q                  ;REP_COUNTER LARGER THAN 30
0216  8B DF                        MOV     BX,DI                            ;THEN MOVE THE KEY NEEDED TO REPEAT
                                                                            ;INTO KEY_Q.
0218  E8 0266 R                    CALL    CHK_BUF
021B  73 03                        JNC     CAL_MV_DATA                      ;CHECK IF KEY_Q IS FULL.
021D  EB 09 90                     JMP     REN_NMI

0220                       CAL_MV_DATA:
0220  E8 0278 R                    CALL    MV_DATA
0223  EB CE                        JMP     SCAN1

0225                       CAL_NO_KEY:
0225  E8 0285 R                    CALL    NO_KEY                           ;NO KEY ENTERED, CLEAR ALL BUFFERS AND
                                                                            ;COUNTERS TO ZEROS.
0228                       REN_NMI:
0228  FB                           STI
0229  E8 0240 R                    CALL    POP_R
022C  CF                           IRET
022D                       KEY_NMI ENDP
                           ;::::::::::::::::::::::::::::::::::::::::::::::::
                                   ;:                                              ::
                                   ;:        REGISTER PUSHING & POPPING ROUTINE    ::
                                   ;:                                              ::
                                   ;:::::::::::::::::::::::::::::::::::::::::::::::::

 Ø22D                              PUSH_R  PROC     NEAR                              ;PUSH ALL REGISTER
 Ø22D   2E: 8F Ø6 Ø2FA R                   POP      CS:IP_MEM
 Ø232   5Ø                                 PUSH     AX
 Ø233   53                                 PUSH     BX
 Ø234   51                                 PUSH     CX
 Ø235   52                                 PUSH     DX
 Ø236   57                                 PUSH     DI
 Ø237   56                                 PUSH     SI
 Ø238   1E                                 PUSH     DS
 Ø239   Ø6                                 PUSH     ES
 Ø23A   2E: FF 36 Ø2FA R                   PUSH     CS:IP_MEM
 Ø23F   C3                                 RET
 Ø24Ø                              PUSH_R  ENDP
                                   ;
 Ø24Ø                              POP_R   PROC     NEAR                              ;POP ALL REGISTER
 Ø24Ø   2E: 8F Ø6 Ø2FA R                   POP      CS:IP_MEM
 Ø245   Ø7                                 POP      ES
 Ø246   1F                                 POP      DS
 Ø247   5E                                 POP      SI
 Ø248   5F                                 POP      DI
 Ø249   5A                                 POP      DX
 Ø24A   59                                 POP      CX
 Ø24B   5B                                 POP      BX
 Ø24C   58                                 POP      AX
 Ø24D   2E: FF 36 Ø2FA R                   PUSH     CS:IP_MEM
 Ø252   C3                                 RET
 Ø253                              POP_R   ENDP
                                   ;:::::::::::::::::::::::::::::::::::::::::::::::::
                                   ;:                                              ::
                                   ;:        COMPARE THE OLD_KEY_BUF WITH THE      ::
                                   ;:        KEY JUST TYPED IN AND STORED IN       ::
                                   ;:        THE AL REGISTER.                      ::
                                   ;:                                              ::
                                   ;:::::::::::::::::::::::::::::::::::::::::::::::::

 Ø253                              COMP    PROC     NEAR                              ;
 Ø253   1E                                 PUSH     DS                                ;
 Ø254   Ø7                                 POP      ES                                ;Z=1 AL IS ALREADY FOUND
 Ø255   8B ØE ØØ21 R                       MOV      CX, OLD_NO_FLG                    ;Z<>1 AL IS A NEW KEY
 Ø259   BF ØØ15 R                          MOV      DI,OFFSET OLD_KEY_BUF
 Ø25C   F2/ AE                             REPNZ    SCASB                             ;THE SYSTEM DEFAULT DIRECTION FLAG IS
 Ø25E   C3                                 RET                                        ;SET TO ZERO, SO THE DI REGISTER WILL
 Ø25F                              COMP    ENDP                                       ;BE INCREMENTED BY ONE EACH TIME.
                                   ;
 Ø25F                              CAPS    PROC     NEAR
 Ø25F   BØ FF                              MOV      AL,ØFFH                           ;INVERSE CAPS_LOCK FLAG.
 Ø261   3Ø Ø6 ØØ2D R                       XOR      CAPS_LOCK,AL
 Ø265   C3                                 RET
 Ø266                              CAPS    ENDP
                                   ;
```

```
                                        ;THE DI POINTS AT THE OFFSET OF A CORRESPONDING BUFFER.
                                        ;CHECK IF A BUFFER OR QUEUE IS FULL.
                                        ;
      0266                              CHK_BUF PROC    NEAR
      0266  1E                                  PUSH    DS
      0267  07                                  POP     ES
      0268  50                                  PUSH    AX
      0269  B9 000A                             MOV     CX,0AH                  ;CX=10, AL=0
      026C  B0 00                               MOV     AL,0                    ;REPEATEDLY COMPARE THE AL WITH A BUFFER
      026E  F2/ AE                              REPNE SCASB                     ;FOR 10 TIMES OR UNTIL ZF=1, ONE BYTE PER T
                                IME.
                                                                                ;AFTER SCANNING IS SUCCESSFULLY FINISHED, T
                                HE DI
                                                                                ;WILL ALWAYS POINT AT THE BYTE OF MEMORY NE
                                XT TO
                                                                                ;THE ONE CONTAINING ZERO VALUE, e.g.,
                                                                                ;
                                                                                ;                          ┌────── DI
                                                                                ;   0    1    2    3    4    5 ...
                                                                                ;                          │
                                                                                ;                          ▼
                                                                                ; ┌────┬────┬────┬────┬────┬─────────
                                                                                ; │ ?? │ ?? │ ?? │ ?? │ 00 │ .....
                                                                                ; └────┴────┴────┴────┴────┴─────────
                                                                                ;
                                                                                ;           A BUFFER
                                                                                ;
      0270  E3 03                               JCXZ    CAL_BEP                 ;IF A BUFFER IS FULL, JUMP TO ROUTINE
      0272  F8                                  CLC                             ;CAL_BEP.
      0273  58                                  POP     AX
      0274  C3                                  RET

      0275                              CAL_BEP:
      0275  58                                  POP     AX                      ;BUFFER OR QUEUE IS FULL, SET THE CARRY FLA
                                G.
      0276  F9                                  STC
      0277  C3                                  RET
      0278                              CHK_BUF ENDP
                                        ;
             CONTAINING ZER             ;THE DI ALWAYS POINTS AT THE BYTE OF MEMORY NEXT TO THE ONE
                                O VALUE
                                        ;
      0278                              MV_DATA PROC    NEAR
      0278  86 65 FF                    MV:     XCHG    [DI-1],AH               ;EXCHANGE THE BYTE OF MEMORY CONTAINING
      027B  88 25                               MOV     [DI],AH                 ;ZERO VALUE WITH THE UNCERTAIN CONTENTS
      027D  4F                                  DEC     DI                      ;OF THE AH REGISTER FOR THE NEXT
      027E  3B FB                               CMP     DI,BX                   ;INSTRUCTION USE. THE AH IS USED AS A
      0280  77 F6                               JA      MV                      ;TEMPARARY BUFFER FOR DATA TRANSFERMATION.
                                                                                ;THE BX POINTS AT THE OFFSET (START) OF
                                                                                ;A CORRESPONDING BUFFER.
      0282                              STORE_Q:                                ;THE AL CONTAINS AN ASCII CODE TO BE SAVED.
      0282  88 07                               MOV     [BX],AL                 ;AFTER SEVERAL DATA TRANSFERMATION, MOVE
      0284  C3                                  RET                             ;THE AL's CONTENTS INTO THE FIRST BYTE OF
                                                                                ;A CORRESPONDING BUFFER.
      0285                              MV_DATA ENDP
                                        ;
```



```
0285                      NO_KEY  PROC    NEAR
0285  FE 06 002A R                INC     NO_KEY_COUNTER
0289  80 3E 002A R 03             CMP     NO_KEY_COUNTER,3                ;IF NO_KEY_FLG>=3, CLEAR NEW_KEY_BUF
028E  72 34                       JB      NO_KEY_1                        ;AND OLD_KEY_BUF TO ZERO. OTHERWISE,
                                                                          ;JUMP TO ROUTINE NO_KEY_1.
0290  B0 00                       MOV     AL,0
0292  B9 000A                     MOV     CX,10
0295  BF 000A R                   MOV     DI,OFFSET NEW_KEY_BUF
0298  BE 0015 R                   MOV     SI,OFFSET OLD_KEY_BUF
029B                      NO_1:
029B  88 05                       MOV     [DI],AL                         ;CLEAR NEW_KEY_BUF AND OLD_KEY_BUF .
029D  88 04                       MOV     [SI],AL
029F  46                          INC     SI
02A0  47                          INC     DI
02A1  E2 F8                       LOOP    NO_1
02A3  C7 06 0024 R 0000           MOV     REP_COUNTER,0                   ;CLEAR ALL THE COUNTERS USED.
02A9  C6 06 0023 R 00             MOV     CAPS_COUNTER,0
02AE  C6 06 0029 R 00             MOV     CTRL_P_COUNTER,0
02B3  C7 06 001F R 0000           MOV     NEW_NO_FLG,0
02B9  C7 06 0021 R 0000           MOV     OLD_NO_FLG,0
02BF  C6 06 002B R 00             MOV     SPECIAL,0
02C4                      NO_KEY_1:                                       ;NO_KEY_COUNTER <3, BUFFERS AND COUNTERS WI
                     LL
                                                                          ;BE LEFT AS WHAT THEY ARE NOW.
02C4  C3                          RET
02C5                      NO_KEY  ENDP
                          ;
                          ; COPY ALL THE ASCII CODES STORED IN THE NEW_KEY_BUF INTO THE OLD_KEY_BUF, AND
                          ; AFTER THAT, CLEAR THE NEW_KEY_BUF TO ZEROS.
                          ;
02C5                      TRANSFER        PROC    NEAR
02C5  8B 0E 001F R                MOV     CX,NEW_NO_FLG
02C9  BF 000A R                   MOV     DI,OFFSET NEW_KEY_BUF
02CC  BE 0015 R                   MOV     SI,OFFSET OLD_KEY_BUF
02CF                      T1:
02CF  B4 00                       MOV     AH,0
02D1  86 25                       XCHG    AH,BYTE PTR [DI]                ;EXCHANGE THE AH WITH A CERTAIN BYTE OF
                                                                          ;THE NEW_KEY_BUF. AFTER THAT, CLEAR A
                                                                          ;BYTE OF THE NEW_KEY_BUF.
02D3  88 24                       MOV     [SI],AH                         ;MOVE TO OLD_KEY_BUF
02D5  46                          INC     SI
02D6  47                          INC     DI
02D7  E2 F6                       LOOP    T1                              ;ITERATE ROUTINE TI FOR THE NUMBER OF TIMES
                                                                          ;STORED IN THE CX REGISTER.
02D9  A0 001F R                   MOV     AL,BYTE PTR NEW_NO_FLG          ;MOVE NEW_NO_FLG TO OLD_NO_FLG
02DC  A2 0021 R                   MOV     BYTE PTR OLD_NO_FLG,AL
02DF  C6 06 001F R 00             MOV     BYTE PTR NEW_NO_FLG,0           ;CLEAR  NEW_NO_FLG
02E4  C3                          RET
02E5                      TRANSFER        ENDP
                          ;::::::::::::::::::::::::::::::::::::::::::::::::
                          ;:                                            ::
                          ;:      GET AN ASCII CODE FROM THE KEY QUEUE  ::
                          ;:      IF QUEUE IS EMPTY, ROUTINE WILL LOOP  ::
                          ;:      UNTILL A KEY IS KEYED IN              ::
                          ;:                                            ::
                               ;::::::::::::::::::::::::::::::::::::::::::::::::

02E5                           GET_KEY PROC    NEAR
0202E5                         WAIT_KEY_DN:
0202E5  1E                             PUSH    DS
0202E6  07                             POP     ES
0202E7  BF 0000 R                      MOV     DI,OFFSET KEY_Q
0202EA  B0 00                          MOV     AL,0
0202EC  B9 000A                        MOV     CX,0AH
0202EF  F2/ AE                         REPNE SCASB                        ;CHECK IF QUEUE IS EMPTY.
0202F1  83 F9 09                       CMP     CX,9
0202F4  74 EF                          JE      SHORT WAIT_KEY_DN
0202F6  86 45 FE                       XCHG    [DI-2],AL                  ;LOAD A ASCII CODE INTO AL
0202F9  C3                             RET
0202FA                         GET_KEY ENDP
                               ;
0202FA  0000                   IP_MEM          DW      0                  ;A WORD MEMORY TO TEMPRARILY STORE THE OFFS
                       ET
                                                                          ;ADDRESS OF AN INSTRUCTION WHICH WAS AUTOMA
                       -
                                                                          ;TICALLY ONTO THE STACK BY THE "CALL"
                                                                          ;INSTRUCTION.
                               ;
0202FC  81 4F 35 09            PTA_TAB DB      81H,'O5',9H                ;DEFINE ALL THE ASCII CODES OF UPPERCASE
020300  36 60 4D 52 31 49              DB      '6`MR1I]P4AG',27H,';E2N'   ;CHARACTERS ON THE KEYBOARD.
        5D 50 34 41 47 27
        3B 45 32 4E
020310  3D 2D 44 5A 48 5B              DB      '=-DZH[9S37/.XWJ'          ;EACH CODE CORRESPONDS TO A POSITION CODE.
        39 53 33 37 2F 2E
        58 57 4A
02031F  0D 4C 46 1B 38 08              DB      0DH,'LF',1BH,'8',08H       ; e.g., 81H TO 1, 'O' TO 2,
020325  30 43 51 59 82 4B              DB      '0CQY',82H,'KT\U '         ; '5' TO 3 ,AND SO ON.
        54 5C 55 20
02032F  2C 56 20 42                    DB      ',V B'


020333  83 6F 25 09            SHIFT   DB      83H,'o%',9H                ;DEFINE ALL THE ASCII CODES OF LOWERCASE
020337  5E 7E 6D 72 21 69              DB      '^~mr!i}p$ag":e@'          ;CHARACTERS ON THE KEYBOARD.
        7D 70 24 61 67 22
        3A 65 40
020346  6E 2B 5F 64 7A 68              DB      'n+_dzh{(s#&?>xwj'
        7B 28 73 23 26 3F
        3E 78 77 6A
020356  0D 6C 66 1B                    DB      0DH,'lf',1bh
02035A  2A 08 29 63 71 79              DB      '*',08h,')cqy',84H
        84
020361  6B 74 7C 75 20 3C              DB      'kt|u <v b'
        76 20 62
02036A                         CODE    ENDS
                                       END     KEY_NMI
```

```
Segments and groups:

                N a m e                 Size    align   combine class

CODE . . . . . . . . . . . . . . .     036A    PARA    PUBLIC  'CODE'
DATA . . . . . . . . . . . . . . .     002E    PARA    PUBLIC  'DATA'

Symbols:

                N a m e                 Type    Value   Attr

AGAIN. . . . . . . . . . . . . . .     L NEAR  001D    CODE
AND_9FH. . . . . . . . . . . . . .     L NEAR  00F5    CODE
BUF_OR_Q_FULL. . . . . . . . . . .     L NEAR  01AD    CODE
CAL_BEP. . . . . . . . . . . . . .     L NEAR  0275    CODE
CAL_CAPS . . . . . . . . . . . . .     L NEAR  009F    CODE
CAL_MV_DATA. . . . . . . . . . . .     L NEAR  0220    CODE
CAL_NO_KEY . . . . . . . . . . . .     L NEAR  0225    CODE
CAPS . . . . . . . . . . . . . . .     N PROC  025F    CODE    Length =0007
CAPS_COUNTER . . . . . . . . . . .     L BYTE  0023    DATA
CAPS_LOCK. . . . . . . . . . . . .     L BYTE  002D    DATA
CHECK_CTRL_P . . . . . . . . . . .     L NEAR  00D6    CODE
CHK_ALT. . . . . . . . . . . . . .     L NEAR  00F7    CODE
CHK_ALT1 . . . . . . . . . . . . .     L NEAR  0059    CODE
CHK_BUF. . . . . . . . . . . . . .     N PROC  0266    CODE    Length =0012
CHK_CAPS . . . . . . . . . . . . .     L NEAR  0104    CODE
CHK_OLD_NO . . . . . . . . . . . .     L NEAR  0129    CODE
CHK_OLD_NO_1 . . . . . . . . . . .     L NEAR  0182    CODE
CHK_REP_COUNTER. . . . . . . . . .     L NEAR  01F9    CODE
CHK_SHIFT. . . . . . . . . . . . .     L NEAR  004D    CODE
COMP . . . . . . . . . . . . . . .     N PROC  0253    CODE    Length =000C
CTRL_P_1 . . . . . . . . . . . . .     L NEAR  00E6    CODE
CTRL_P_COUNTER . . . . . . . . . .     L BYTE  0029    DATA
FILL_0 . . . . . . . . . . . . . .     L NEAR  0155    CODE
FILL_1 . . . . . . . . . . . . . .     L NEAR  0174    CODE
FILL_IN_NEW_BUF. . . . . . . . . .     L NEAR  014E    CODE
FILL_IN_NEW_BUF_AND_Q. . . . . . .     L NEAR  0188    CODE
FIND_NEXT_KEY. . . . . . . . . . .     L NEAR  007D    CODE
FIRST_REP. . . . . . . . . . . . .     L NEAR  0213    CODE
GET_KEY. . . . . . . . . . . . . .     N PROC  02E5    CODE    Length =0015
INC_1. . . . . . . . . . . . . . .     L NEAR  016A    CODE
IPD_PORT . . . . . . . . . . . . .     Number  01C0
IP_MEM . . . . . . . . . . . . . .     L WORD  02FA    CODE
KCOL . . . . . . . . . . . . . . .     L NEAR  0062    CODE
KCOL1. . . . . . . . . . . . . . .     L NEAR  0068    CODE
KEY_DN . . . . . . . . . . . . . .     L NEAR  008B    CODE
KEY_DN_1 . . . . . . . . . . . . .     L NEAR  00AB    CODE
KEY_NMI. . . . . . . . . . . . . .     F PROC  0000    CODE    Length =022D
KEY_Q. . . . . . . . . . . . . . .     L BYTE  0000    DATA    Length =000A
KROW . . . . . . . . . . . . . . .     L NEAR  0079    CODE
LAST_KEY_FLG . . . . . . . . . . .     L BYTE  0028    DATA
MV . . . . . . . . . . . . . . . .     L NEAR  0278    CODE
MV_DATA. . . . . . . . . . . . . .     N PROC  0278    CODE    Length =000D
NEW_KEY_BUF. . . . . . . . . . . .     L BYTE  000A    DATA    Length =000B
NEW_NO_FLG . . . . . . . . . . . .     L WORD  001F    DATA
```

```
The Microsoft MACRO Assembler , Version 1.25                Page     Symbols-2
                                                            12-21-84

NMI_IN . . . . . . . . . . . . . .          L NEAR  0024     CODE
NOCTRL . . . . . . . . . . . . . .          L NEAR  00C1     CODE
NOSHIFT. . . . . . . . . . . . . .          L NEAR  00C9     CODE
NO_1 . . . . . . . . . . . . . . .          L NEAR  029B     CODE
NO_KEY . . . . . . . . . . . . . .          N PROC  0285     CODE     Length =0040
NO_KEY_1 . . . . . . . . . . . . .          L NEAR  02C4     CODE
NO_KEY_COUNTER . . . . . . . . . .          L BYTE  002A     DATA
OLD_KEY_BUF. . . . . . . . . . . .          L BYTE  0015     DATA     Length =000A
OLD_NO_FLG . . . . . . . . . . . .          L WORD  0021     DATA
OPD_PORT1. . . . . . . . . . . . .          Number  0180
OPD_PORT2. . . . . . . . . . . . .          Number  0160
POP_R. . . . . . . . . . . . . . .          N PROC  0240     CODE     Length =0013
PTA_TAB. . . . . . . . . . . . . .          L BYTE  02FC     CODE
PTR_FLG. . . . . . . . . . . . . .          L BYTE  002C     DATA
PUSH_R . . . . . . . . . . . . . .          N PROC  022D     CODE     Length =0013
REN_NMI. . . . . . . . . . . . . .          L NEAR  0228     CODE
REP_COUNTER. . . . . . . . . . . .          L WORD  0024     DATA
REP_COUNTER1 . . . . . . . . . . .          L WORD  0026     DATA
SCAN . . . . . . . . . . . . . . .          L NEAR  002E     CODE
SCAN0. . . . . . . . . . . . . . .          L NEAR  01ED     CODE
SCAN1. . . . . . . . . . . . . . .          L NEAR  01F3     CODE
SCAN_OUT . . . . . . . . . . . . .          L NEAR  01B4     CODE
SHIFT. . . . . . . . . . . . . . .          L BYTE  0333     CODE
SHIFT_CAPS_LOCK. . . . . . . . . .          L NEAR  011F     CODE
SPECIAL. . . . . . . . . . . . . .          L BYTE  002B     DATA
STORE_Q. . . . . . . . . . . . . .          L NEAR  0282     CODE
T1 . . . . . . . . . . . . . . . .          L NEAR  02CF     CODE
TRANSFER . . . . . . . . . . . . .          N PROC  02C5     CODE     Length =0020
WAIT_KEY_DN. . . . . . . . . . . .          L NEAR  02E5     CODE

Warning Severe
Errors  Errors
0       0
```

```
 Symbol Cross Reference                 (# is definition)       Cref-1

AGAIN. . . . . . . . . . . . . . .   78#   82
AND_9FH. . . . . . . . . . . . . .  201   218#

BUF_OR_Q_FULL. . . . . . . . . . .  279   293   3Ø5   311   317#  342

CAL_BEP. . . . . . . . . . . . . .  457   462#
CAL_CAPS . . . . . . . . . . . . .  159   163#
CAL_MV_DATA. . . . . . . . . . . .  364   367#
CAL_NO_KEY . . . . . . . . . . . .  325   371#
CAPS . . . . . . . . . . . . . . .  166   429#  433
CAPS_COUNTER . . . . . . . . . . .   32#  157   164   5ØØ
CAPS_LOCK. . . . . . . . . . . . .   45#  232   431
CHECK_CTRL_P . . . . . . . . . . .  187   199#
CHK_ALT. . . . . . . . . . . . . .  192   197   221#
CHK_ALT1 . . . . . . . . . . . . .  113   118#
CHK_BUF. . . . . . . . . . . . . .  278   292   3Ø4   31Ø   341   363   438#  466
CHK_CAPS . . . . . . . . . . . . .  223   225   231#
CHK_OLD_NO . . . . . . . . . . . .  233   237   24Ø   243   247   249   252#
CHK_OLD_NO_1 . . . . . . . . . . .  282   288   296#
CHK_REP_COUNTER. . . . . . . . . .  334   35Ø#
CHK_SHIFT. . . . . . . . . . . . .  1Ø8   111#
CODE . . . . . . . . . . . . . . .   48#   48    5Ø   584
COMP . . . . . . . . . . . . . . .  261   42Ø#  427
CTRL_P_1 . . . . . . . . . . . . .  2Ø4   2Ø7#
CTRL_P_COUNTER . . . . . . . . . .   39#  2Ø2   2Ø8   5Ø1

DATA . . . . . . . . . . . . . . .    2#    2    46    5Ø    5Ø    68    85

FILL_Ø . . . . . . . . . . . . . .  269   275#
FILL_1 . . . . . . . . . . . . . .  265   274   289#
FILL_IN_NEW_BUF. . . . . . . . . .  267   272#
FILL_IN_NEW_BUF_AND_Q. . . . . . .  257   263   3ØØ#
FIND_NEXT_KEY. . . . . . . . . . .  142#  161   168   2Ø6   213   298   315
FIRST_REP. . . . . . . . . . . . .  353   359#

GET_KEY. . . . . . . . . . . . . .   79   54Ø#  552

INC_1. . . . . . . . . . . . . . .  283#  287
IPD_PORT . . . . . . . . . . . . .    8#  1Ø1   133
IP_MEM . . . . . . . . . . . . . .  386   395   4ØØ   4Ø9   554#

KCOL . . . . . . . . . . . . . . .  115   12Ø   122#
KCOL1. . . . . . . . . . . . . . .  126#  15Ø
KEY_DN . . . . . . . . . . . . . .  139   153#
KEY_DN_1 . . . . . . . . . . . . .  156   17Ø#
KEY_NMI. . . . . . . . . . . . . .   49#  378   585
KEY_Q. . . . . . . . . . . . . . .   12#  3Ø8   339   36Ø   544
KROW . . . . . . . . . . . . . . .  138#  146

LAST_KEY_FLG . . . . . . . . . . .   35#  264   268   273   281   3Ø1   323

MV . . . . . . . . . . . . . . . .  471#  475
MV_DATA. . . . . . . . . . . . . .  28Ø   294   3Ø6   312   343   368   47Ø#  482

NEW_KEY_BUF. . . . . . . . . . . .   17#  276   29Ø   3Ø2   329   491   515
NEW_NO_FLG . . . . . . . . . . . .   28#  253   324   336   5Ø2   514   527   529
NMI_IN . . . . . . . . . . . . . .   64    66    83#
```

```
 Symbol Cross Reference              (# is definition)       Cref-2

NOCTRL . . . . . . . . . . . . . .  18Ø    188#
NOSHIFT. . . . . . . . . . . . . .  175    193#
NO_1 . . . . . . . . . . . . . . .  493#   498
NO_KEY . . . . . . . . . . . . . .  372    484#   5Ø8
NO_KEY_1 . . . . . . . . . . . . .  487    5Ø5#
NO_KEY_COUNTER . . . . . . . . . .   4Ø#   326    485    486

OLD_KEY_BUF. . . . . . . . . . . .   22#   266    33Ø    424    492    516
OLD_NO_FLG . . . . . . . . . . . .   3Ø#   255    327    337    423    5Ø3    528
OPD_PORT1. . . . . . . . . . . . .    6#    73     99    131
OPD_PORT2. . . . . . . . . . . . .    7#    96    127

POP_R. . . . . . . . . . . . . . .  16Ø    167    2Ø5    212    297    314    318    376    399#   411
PTA_TAB. . . . . . . . . . . . . .  181    194    559#
PTR_FLG. . . . . . . . . . . . . .   44#   211
PUSH_R . . . . . . . . . . . . . .   84    154    385#   397

REN_NMI. . . . . . . . . . . . . .  32Ø    348    365    374#
REP_COUNTER. . . . . . . . . . . .   33#   285    345    351    499
REP_COUNTER1 . . . . . . . . . . .   34#   354    355    357

SCAN . . . . . . . . . . . . . . .   94#
SCANØ. . . . . . . . . . . . . . .  338    344#
SCAN1. . . . . . . . . . . . . . .  328    346#   352    356    369
SCAN_OUT . . . . . . . . . . . . .  152    322#
SHIFT. . . . . . . . . . . . . . .  189    572#
SHIFT_CAPS_LOCK. . . . . . . . . .  235    245#
SPECIAL. . . . . . . . . . . . . .   43#   1Ø4    1Ø9    114    121    171    178    196    222    234    5Ø4
STORE_Q. . . . . . . . . . . . . .  478#

T1 . . . . . . . . . . . . . . . .  517#   525
TRANSFER . . . . . . . . . . . . .  347    513#   531

WAIT_KEY_DN. . . . . . . . . . . .  541#   549
```

At the start of the example program, the DEFINE assembler directives are used to define the variables used in the program. Here, we are going first to explain some of the critical ones in more detail:

1). OPD_PORT1: A name used by the Assembler to represent the I/O port 180H, which is an output port to the program.

2). OPD_PORT2: A name used by the Assembler to represent the I/O port 160H, which is an output port to the program.

3). IPD_PORT: A name used by the Assembler to represent the I/O port 1C0H, which is an input port to the program.

4). KEY_Q: A buffer (an area of memory), referred to as key queue for storing keyboard inputs. It has 10 bytes of memory.

5). NEW_KEY_BUF:

A buffer whose function is somewhat similar to key queue. It stores one up to 11 latest keyboard inputs after a complete scan-out. Then, data in this buffer will be always transferred to KEY_Q and OLD_KEY_BUF, respectively, after a complete scan-out. Thus, the program can determine the key just being entered is a new key or a repeated key by making a comparison with the values in both variables NEW_KEY_BUF and OLD_KEY_BUF.

6). NEW_NO_FLG:

A word variable storing the number of latest keyboard inputs. Its contents will be transferred to the variable OLD_NO_FLG after each scan-out.

7). OLD_KEY_BUF:

A 10-byte buffer storing the keyboard inputs that were keyed in at the last scan-out operation.

8). OLD_NO_FLG:

A word variable storing the number of last keyboard inputs.

9). CAPS_COUNTER:

A byte variable whose contents will be automatically incremented by one if one holds down the CAP LOCK key continuosly. When it equals to one, the program will enable the CAP LOCK key; otherwise, disable that key.

10). REP_COUNTER:

A word variable used to determine if a key same as the last key should be repeated. If one holds down a key continuosly, the program will recognize its subsequent keys as repeated keys after scanning it for 30 times; the program will perform 30 times of scan-out operations to accept the first repeated key.

11). REP_COUNTER1:

A word variable used to determine if a key same as the last key should be repeated. If one still holds down a same key after the program has displayed its corresponding character twice on the screen, the program will keep going to recognize its subsequent keys as repeated keys every 4 times of scan-out.

12). LAST_KEY_FLG:

A byte variable used as an internal flag to set the last key one entered from the keyboard after the program completes a scan-out operation. If it contains the value of FFH, then the program will know that the key one just entered is the same as the last key of last scan-out.

13). CTRL_P_COUNTER:

A byte variable whose contents will be automatically incremented by one if one holds down both the keys CTRL and P continuosly. The code CTRL_P is used to enable the printer driver. When this variable's content equals to one, the program will accept this code; otherwise, it won't accept.

14). NO_KEY_COUNTER:

A byte variable which is initialized to 0. How does the program assume that there is no more key entered from the keyboard after a scan-out operation? What the program does for this purpose is to perform three scan-out operations successively. After completing each scan-out, the variable is incremented by one. If no key is definitely sighted after performing three scan-out operations, the program assumes, based on the fact that the value of the variable has been incremented to three, that there is no more key entered.

15). SPECIAL:	A byte variable which has three basic usages. The program uses its first bit (bit Ø) to determine if the SHIFT key is pressed, its second bit (bit 1) to determine if the ALT key is pressed, and its third bit (bit 2) to determine if the CTRL key is pressed.
16). PTR_FLG:	A byte variable used as a flag to determine if the printer drive shall be enabled or not. It is the only variable used in the keyboard program whose contents will be needed by some external routines associated with the printer driver. Its contents are either FFH or ØH. Thus, to our program, it is an output.
17). CAPS_LOCK:	A byte variable used as an internal flag to determine if the CAP_LOCK key shall be enabled or not. If it contains the value FFH, then the CAP_LOCK key will work.

Now, please take a look at the end of the example program. You can see two variables, named PTA_TAB and SHIFT respectively. As with all the keyboards designed by other manufacturers, every key on the keyboard of the MPF-I/88 has its own corresponding ASCII code. We define the ASCII code of each key in the keyboard program by two Define-Byte look-up tables whose names are stated above.

The first one, PTA_TAB, (Position code to ASCII code Table) defines the ASCII codes of the lowercase keys. The second, SHIFT, (position code of the uppercase key to ASCII code Table) defines the ASCII codes of the uppercase keys.

Those characters in both tables PTA_TAB and SHIFT with two apostrophys tells the assembler to generate standard ASCII codes. Others, such as 81H, 9H, 27H, etc., are our own codes which indicate special keys. Refer to Tables 5-1 and 5-2 Conversion Tables for each key's position code and its corresponding ASCII code.

Before analyzing the program, let us first examine its structure and the function of each routine. The keyboard program totally consists of ten major routines, including the main program KEY_NMI. They are:

Name	Functions
1. KEY_NMI	The main program which is composed of two parts. Part one stores the starting address of the core keyboard program into two memory locations addressed by Ø8 and ØA, which are part of the

interrupt service routine address table, and initializes the keyboard hardware interface by sending out a value of ØFFH to the I/O port Ø18ØH.

Part two is an infinite loop labelled AGAIN that reads in keyboard input and displays it on the LCD screen. The other components of the program is our major keyboard program starting with statements labelled NMI_IN.

2. PUSH_R	A subroutine for pushing the status of all registers onto the stack.
3. POP_R	A subroutine for restoring all register from the stack.
4. COMP	A subroutine which determines if the key just entered is a new depressed one.
5. CAPS	A subroutine for initializing the variable CAPS_LOCK to the value FFH as stated before.
6. CHK_BUF	A subroutine for checking if any buffer used in the program is full of valid data any time it is needed to test.
7. MV_DATA	A subroutine for moving data in the AL register (a key just entered) into one of both buffers NEW_KEY_BUF and KEY_Q, depending on the situation.
8. NO_KEY	A decision-making subroutine for determining if there is no more key entered from the keyboard after a complete scan-out and initializing to zero all the variables used as counters in the program if the program recognizes that there is no key entered after performing two complete scan-out operations.
9. TRANSFER	A subroutine for moving current data in the buffer NEW_KEY_BUF into the buffer OLD_KEY_BUF and then clearing the buffer NEW_KEY_BUF to zeros.
1Ø. GET_KEY	A subroutine for moving the "first-in" data in the buffer KEY_Q into the AL register in order to display it on the screen.

We can now look at the main program KEY_NMI. The program first has to set the starting address of our own keyboard program (starting with the instruction labelled NMI_IN) into proper entries addressed by Ø8H and ØAH of the interrupt-service- outine vector table.

Remember that the program will be invoked every 15 milliseconds. Once the vector table is modified, the program enables the fifth bit (bit 4) of the I/O port Ø18ØH to allow external interrupt sources to interrupt pin NMI of the 8Ø88 by outputing the value ØFFH through the DX register.

The second part routine labelled AGAIN is a finite loop that calls the routine GET_KEY to obtain characters input from the keyboard. Each character so received is echoed on the screen by the instruction INT 9. Please refer to the chapter on useful subroutines of the MPF-I/88 User's Manual for the usage of INT 9.

If we strike a key while this loop is running, an NMI interrupt will occur. This will cause our keyboard program starting with the instruction labelled NMI_IN to be activated.

Let's go down to the routine labelled NMI_IN. One thing you have to keep in mind is that if you program is associated with (or called by) someone's program, you have to save onto the stack the current status of all registers in the CPU that the calling program just produced before executing your own program; then at the end of program, restore them back to their original respective register. Thus, you will not destroy them during the course of the program's execution. Based on the rule of programming, Statements labelled NMI_IN start with the instruction CALL PUSH_R performing what we stated just now - push all the registers onto the stack.

Routines from SCAN through CHK_ALT1 are used to check if the three keys of column KC-11 (CTRL, SHIFT, and ALT keys) are entered. If the key entered is of CTRL key, then bit 3 of the variable SPECIAL will be set to 1 for future use somewhere else in the program. This is true for the SHIFT and ALT keys.

Statements, starting from the instruction labelled KCOL and ending with the last instruction JMP SCAN_OUT of the routine labelled FIND_NEXT_KEY, begin to scan the keyboard matrix from column KC-1Ø by sending the value ØFFH to I/O port Ø16ØH and the value ØFBH to I/O port Ø18ØH, respectively.

After scanning up each culumn, the equivalent binary number of ØFBFFH will be shifted one digit to the right in order to scan the next column. This is done by the instruction SAR AX,1 contained in the routine FIND_NEXT_KEY.

If the program recognizes that there is a key entered from the keyboard, control of execution will turn to the instruction labelled KEY_DN (key down). Remember that the DI register is used as a count for the position code of a key in our program. It will be automatically incremented by one after a key of the matrix (KC-10 x KC-0) has been scanned out.

Routines from KEY_DN through CTRL_P_1 first check if the key entered is the CAP LOCK key (a key to set the uppercase character) whose position code in our program is 50. How can we obtain the ASCII code of a key? Because each key has its own position code stored in the DI register, we use it as a displacement (namely, offset) between the beginning of table PTA_TAB and the ASCII character desired. And then we again find out the starting address of table PTA_TAB by the instruction MOV BX,offset PTA_TAB. Finally, by adding the BX and DI, we get the corresponding ASCII code of a key and store it in the AL register again.

Let us now look at the routine AND_9FH. This "AND" instruction is used to obtain the ASCII code of a certain control character. For instance, the hex value of character A is 41H. And performing this AND logic operation will result in the ASCII code (01H) of the control character CTRL_A.

The default image of characters shown on the LCD screen of the MPF-I/88 is in upper case (capital letter). If one enters any alphabetic character with the CAP LOCK key, the program will accept it as lowercase characters. Try to trace the routines CHK_CAPS and SHIFT_CAPS_LOCK, you will understand the meaning of those statements.

The statements starting from the instruction labelled CHK_OLD_NO seem to be a little difficult to trace, for there are some factors we have to take into account. For example, one might strike more than one key (up to ten keys) at a time without releasing them. What the program should have to do is to accept them and display them on the screen, then repeat displaying the character on the screen which was depressed last. This is accomplished by routines labelled CHK_OLD_NO, FILL_IN_NEW_BUF, FILL_0, FILL_1, and CHK_OLD_NO_1.

Assuming that a valid key has been struck, we now have its ASCII code in the AL register. We must place this byte onto the key queue so that it is available to the GET_KEY routine. One more thing we have to do is to place this byte into buffer NEW_KEY_BUF as well for next scanning comparison. This is accomplished by the routine labelled FILL_IN_NEW_FUF_AND_Q.

Now, let us look at routines from SCAN_OUT through REN_NMI. Each time the program completes a scanning operation from KC-11 to KC-0, control of the pregram execution will be tranfered to here to initialize some counters and internal flags. Finally, the program returns control of the execution to the CPU to end up itself by executing the instruction IRET which is located at the end of the REN_NMI routine.

Recall that the Instruction Pointer (IP) is used to tell the CPU the address of the next instruction to be executed. When you use the CALL instruction, the CPU will save the IP on the top of the stack for jumping back, and then control of the program execution will be transferred to the subroutine desired. Let us now look at the PUSH_R routine and see its first instruction POP CS:IP_MEM. This instruction is used to pop off the stack into memory somewhere in the Code Segment the IP which the CPU just pushed onto the stack.

At this point, you might ask why we put this instruction here. This is because the PUSH_R routine is used to push the current status of registers required onto the stack. If we don't write this instruction in this routine, the IP will be forced down to the bottom of the stack; thus, once this PUSH_R routine is completed, the RET instruction can not cause control of the program to go back to the address that the CALL instruction saved earlier, for the address stored on the top of the stack is not of the origianl contents of the IP. Therefore, in order to ensure that the IP is always on the top of the stack, we first pop it off the stack, and then push it again onto the top of the stack after pushing all the register we required.

The rest of the program we do not explain are considered not difficult for you to trace. Try to trace all the program, then you can experience many skills in programming in 8088 assembly language.

Appendix A

Introduction to 8088 Assembly Language

1. Data Transfer Instructions

Data transfer instructions are used to move data from a specified point to another. The data that is transferred may be in groups of 8 bits or 16 bits.

Most data transfer instructions have two operands, such as MOV. The first operand is called the destination operand, in which the result of the operation is stored. The second operand is the source operand, which stores the data before transfer. Some of data transfer instructions only have one operand, such as POP and PUSH. The only operand can be a source operand or a destination operand.

Data transfer instructions that we will introduce to you below includes: MOV (move), PUSH, POP, XCHG (exchange), IN, OUT (input/output ports), and XLATB (translate). Each instruction is described in parts as follows: Description, Flag register bits affected, Syntax, and Example.

1. Data Transfer Instructions

Data transfer instructions are used to move data from a specified point to another. The data that is transferred may be in groups of 8 bits or 16 bits.

Most data transfer instructions have two operands, such as MOV. The first operand is called the destination operand, in which the result of the operation is stored. The second operand is the source operand, which stores the data before transfer. Some of data transfer instructions only have one operand, such as POP and PUSH. The only operand can be a source operand or a destination operand.

Data transfer instructions that we will introduce to you below includes: MOV (move), PUSH, POP, XCHG (exchange), IN, OUT (input/output ports), and XLATB (translate). Each instruction is described in parts as follows: Description, Flag register bits affected, Syntax, and Example.

MOV

Description: MOV = move

Move a byte or a word from the source operand to the destination operand. The source operand can be memory, register or an immediate value. The destination operand can be memory or register.

Flag registers affected: none.

Syntax:

```
        MOV reg,mem/reg

        MOV mem/reg,reg

        MOV mem/reg,numb
```

Example 1:

```
        MOV DX,3        ;move 3 to DX
        MOV AX,Ø        ;initialize AX to Ø
        MOV AX,DX       ;move the content of DX to AX
```

PUSH

Description: PUSH = push

Decrease SP (the Stack Pointer) by 2, and then store a word from the source operand to the current top of the stack that SP points to.

Flag registers affected: none.

Syntax:

```
        PUSH reg/mem
```

Example 1:

```
        PUSH BX         ;store the contents of BX to the stack
```

Example 2:

```
        PUSH  [123]     ;store the  contents of the memory location
                        ;in the  DS that is  addressed by the value
                        ;123H to the stack
```

POP

Description: POP = pop

Remove the word at the top of stack that SP points to the destination operand, and then increase the SP by 2 to point to the new top of the stack.

Flag registers affected: none.

Syntax:

```
        POP reg/mem16
```

Example 1:

```
        POP DX          ;store the word at the top of stack to
                        ;DX
        MOV AX,DX       ;store contents
```

Example 2:

```
        POP [123]       ;pop off the stack to the memory  location
                        ;in the DS which is addressed by the value
                        ;123H
```

XCHG

Description: XCHG =exchange

Swap the contents of the source and the destination operands.

Flag registers affected: none.

Syntax:

```
        XCHG mem/reg,reg
```

Example 1:

```
        MOV  AX,3       ;move 3 to AX
        MOV  BX,5       ;move 5 to BX
        XCHG AX,BX      ;move  the  contents of AX to  BX  and
                        ;move the contents of BX to AX
```

IN

Description: IN = input port

Transfer a byte or a word from an input port to AL. The port number can be an immediate constant or can be stored in the DX.

Flag register bits affected: none

Syntax:

```
        IN AL/AX,port
```

Example 1:

```
        MOV AX,Ø        ;Initialize AX to Ø.
        MOV DX,1A3      ;Move the I/O port address 1A3 into the DX
                        ;register.  The  content of current  cursor
                        ;position  can  be read from the  I/O  port
                        ;1A3.
        IN AL,DX        ;Read  the content of cursor position  from
                        ;I/O port.
        INT 7           ;Return control to the monitor program.
```

After the program has been executed, the AX register will contain ØØ2Ø and the DX register will contain Ø1A3.

OUT

Description: OUT = output port

This instruction outputs a byte or a word from AL or AX to an output port. The port number may be an immediate value or may be placed in the DX.

Flag register bit affected: none

Syntax:

```
        OUT port,AL/AX
```

Example 1:

```
        MOV AL,41       ;Move ASCII code 41H ('A') to AL.
        MOV DX,1A1      ;Move  I/O  port address 1A1  into  the  DX
                        ;register  so that the contents of DX point
                        ;to the output port,  which is used by  the
                        ;LCD.
        OUT DX,AL       ;Write data contained in the AL onto LCD.
        INT 7           ;Return control to the monitor program.
```

The character 'A' will be displayed on the LCD after executing the program.

XLAT

Description: XLAT = translate

This instruction is used to translate characters from one code to another, such as, ASCII to EBCDIC or vice versa. It replaces a byte in the AL with a byte from a 256-byte, user-coded translation table. BX is usually assumed as the beginning of the translation table. AL is regarded as the offset. The instruction plus BX and AL and then move the content of the result address to AL.

Flag register bits affected: none

Syntax:

```
        XLAT
```

Example 1:

```
        MOV AL,ØF
        MOV BX,4Ø
        MOV BYTE[4F],11
        XLAT                 ;change the content of AL from F to 11
        INT 7
```

After the execution, the AL will be loaded with 11H which is originally stored on the memory location 4F.

2. Arithmetic Instructions

In this section, we will describe the arithmetic instructions as follows: ADD (addition), INC (increment), SUB (subtract), DEC (decrement), NEG (negate), CMP (compare), MUL (multiply), and DIV (divide).

The arithmetic instructions provide the following four basic operations: addition, subtraction, multiplication and division. You can use these instructions to manipulate the following types of numbers: unsigned binary, signed binary, unsigned packed decimal, and unsigned unpacked decimal.

The contents of the flag register can be 1s or Øs. Like all registers, it is a 16-bit register. Nine of the 16 bits are used independently as flags and are used to reflect different kinds of results from arithmetic operations. Seven bits are unused on the 8Ø88. Some of the more important flag bits are described below. A flag is set if it is 1. It is clear if it is Ø.

CF (carry flag) is set if there is a carry out of the most singificant bit or borrow into the most significant bit. Otherwise, it is cleared.

PF (parity flag) is set if the result of an arithmetic operation has an even number of 1-bits. Otherwise, it is cleared. Note that the parity flag only tests byte length data.

AF (auxiliary flag) is set if there is a carry out of bit 3 to bit 4, or a borrow from bit 4 to bit 3. Otherwise, it is cleared. You can use this flag in both 8-bit or 16-bit arithmetic operations.

ZF (zero flag) is set if the result of the operation equals Ø. Otherwise, it is cleared.

SF (sign flag) is set if the result of the operation is less than Ø. It is cleared if the result is larger than or equal to Ø.

OF (overflow flag) is set if the result of the operation is larger than its destination operand.

ADD

Description: ADD = addition

Add the source operand to the destination operand and place the sum in the destination operand. The sum may be a byte or a word.

Flag register bits affected: AF, CF, OF, PF, SF, ZF.

Syntax:

```
        ADD reg,mem/reg

        ADD mem/reg,reg

        ADD mem/reg,numb
```

Example 1:

```
        MOV AX,7
        MOV CX,2
        ADD CX,AX       ;add contents of AX to CX and return the
                        ;result to CX
```

Example 2:

```
        MOV CL,5
        ADD CL,2        ;add  immediate  value 2H  to  CL  and
                        ;return the result to CL
```

INC

Description: INC = increment

Add one to the destination operand, which may be a byte or a word.

Flag register bits affected: AF, OF, PF, SF, ZF.

Syntax:

```
        INC reg/mem
```

Example 1:

```
   ADDRESS       MNEMONICS  OPERANDS         COMMENTS

   ØØ8Ø:ØØØØ       MOV      CX,A             ;move value 1Ø into CX
   ØØ8Ø:ØØØ3       MOV      BYTE[1ØØ],Ø1     ;move  value  Ø1 into  the
                                             ;memory location
   ØØ8Ø:ØØØ8       MOV      DI,1Ø1
   ØØ8Ø:ØØØB       MOV      AL,[1ØØ]         ;move the contents of  the
                                             ;memory location addressed
                                             ;by 1ØØ into AL
   ØØ8Ø:ØØØE       MOV      [DI],AL          ;move the  contents in the
                                             ;AL  into   the    memory
                                             ;location addressed by the
                                             ;contents in DI
   ØØ8Ø:ØØ1Ø       INC      DI               ;add  one to DI and return
                                             ;the result to DI
   ØØ8Ø:ØØ11       INC      BYTE[1ØØ]        ;add   one to  the  memory
                                             ;location addressed by 1ØØ
   ØØ8Ø:ØØ15       LOOP     ØB               ;if CX is not equal to  Ø,
                                             ;jump    to   the   memory
                                             ;location addressed by the
                                             ;value ØB
   ØØ8Ø:ØØ17       INT      7                ;transfer  control  to the
                                             ;monitor program
```

After execution, the memory locations ranging from 1ØØ to 1ØA will be as follows:

1ØØ	ØB
1Ø1	Ø1
1Ø2	Ø2
1Ø3	Ø3
1Ø4	Ø4
1Ø5	Ø5
1Ø6	Ø6
1Ø7	Ø7
1Ø8	Ø8
1Ø9	Ø9
1ØA	ØA

SUB

Description: SUB = subtract

Subtract the source operand from the destination operand, and place the difference into the destination operand. The contents of either operand may be signed or unsigned numbers.

Flag register bits affected: AF, CF, OF, PF, SF, ZF.

Syntax:

```
        SUB reg,mem/reg

        SUB mem/reg,reg

        SUB mem/reg,numb
```

Example 1:

```
        MOV CX,9
        MOV BX,3
        SUB CX,BX       ;subtract  BX  from CX  and return the
                        ;difference to CX
        INT 7
```

After execution, the CX will contain Ø6.

Example 2:

```
        MOV AL,1Ø
        SUB AL,A        ;subtract  1Ø  from AL  and return the
                        ;difference to AL
        INT 7
```

After execution, the AL will contain Ø6.

DEC

Description: DEC = decrement

Subtract one from the destination operand. The operand must be an unsigned binary number, which can be a byte or a word.

Flag register bits affected: AF, OF, PF, SF, ZF.

Syntax:

```
        DEC reg/mem
```

Example 1:

```
        DEC AX          ;subtract one from  AX and  return the
                        ;result to AX
```

Example 2:

```
        DEC BYTE[123]      ;subtract one from the contents of
                           ;memory location 123
```

NEG

Description: NEG = negate

Produce two's complement of the destination operand, that is, reverse the sign of the number.

Flag register bits affected: AF, CF, OF, PF, SF, ZF.

Syntax:

```
        NEG reg/mem
```

Example 1:

```
        MOV AX,Ø
        MOV AL,Ø1         ;move value 1 into AL
        NEG AX            ;change AX to FFFF
        NEG AL            ;change AL to its original value Ø1
        INT 7
```

After execution, the AX will contain FFØ1.

CMP

Description: CMP = compare

Compare two operands by subtracting the source from the destination. Both operands are unchanged since the difference is not placed in the operand. CMP can be followed by any conditional jump instruction. If the destination is greater than the source jump is taken.

Flag register bits affected: AF, CF, OF, PF, SF, ZF.

Syntax:

```
        CMP reg,mem/reg

        CMP mem/reg,reg

        CMP mem/reg,numb
```

Example 1:

```
        CMP BX,CX        ;compare BX with CX
```

Example 2:

```
        CMP BL,Ø2        ;compare BL with Ø2H
```

Example 3:

```
        CMP WORD[7F2],16       ;Subtract value 16H from memory
                               ;location addressed by 7F2 (low byte)
                               ;and 7F3 (high byte), and use the
                               ;result to set the flags. The result
                               ;of this operation is not stored back
                               ;into the specified locations.
```

Suppose memory location 7F2 contains Ø1H and 7F3 contains FFH, the contents of thess two memory locations are not changed after execution.

MUL

Description: MUL = multiply

Multiply source operand by AX or AL. If the source operand is a word, multiply it by AX and return the product in DX and AX. If the source operand is a byte, multiply it by AL and return the product in AH and AL. The operand is unsigned binary numbers.

Flag register bits affected: CF, OF.

AF, PF, SF, ZF undefined.

Syntax:

```
        MUL mem/reg
```

Example 1:

```
        MOV AX,3
        MOV CX,2
        MUL CX          ;multiply AX by the contents of CX
        INT 7
```

AX will contain Ø6H after execution and DX contains ØØ.

DIV

Description: DIV = divide

Divide the dividend by the source operand. If the source operand is a byte, it divides the dividend in AH and AL and then returns the remainder in AH and the quotient in AL. If the source operand is a word, it divides the dividend in DX and AX and then returns the remainder in DX an the quotient in AX.

Flag register bits affected: AF, CF, OF, PF, SF, ZF are undefined.

Syntax:

```
        DIV mem/reg
```

Example 1:

```
        DIV CL          ;CL divides what in AH and AL
```

Example 2:

```
        MOV DX,23
        MOV AX,4
        MOV CX,3E8
        DIV CX          ;Divide ØØ23ØØØ4H by 3E8H
                        ;(divide DX:AX by CX)
        INT 7
```

After execution, the AX will contain Ø8E5H and the DX will contain Ø2FCH.

3. Logical Instructions

The logical instructions include NOT, AND, OR, XOR, TEST, SHL, SHR, RCL, ROL, RCR, and ROR.

Unlike the arithmetic instructions which always regards their operands as numbers, the logical instructions regards their operands as strings of bits. In addition, the logical instructions can operate on a byte or a word operand.

The flags are not affected by the logical NOT. However, AND, OR, XOR and TEST affects the status of the flag register as follows:

CF:	cleared.
OF:	cleared.
AF:	undefined.
PF:	set for even number of 1-bits, clear for odd number of 1-bits.
SF:	depends on the status of the high-bit of the operand.
ZF:	depends on the numeric value of the operand.

NOT

Description:

Form the one's complement of the destination operand. The destination may be a byte or a word.

Flag register bits affected: none.

Syntax:

```
        NOT reg/mem
```

Example 1:

```
        NOT BYTE[100]
```

Suppose memory location 100H contains 80H, after execution, its contents will be changed to 7FH.

AND

Description:

Perform the logical "and" bit by bit between the source operand and the destination operand. The result is stored in the destination.

```
        Ø AND Ø = Ø
        Ø AND 1 = Ø
        1 AND Ø = Ø
        1 AND 1 = 1
```

Flag register bits affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
        AND reg,mem/reg

        AND mem/reg,reg

        AND mem/reg,numb
```

Example 1:

```
        AND CX,ØFF
```

Example 2:

```
        AND AX,BX
```

OR

Description: OR = inclusive OR

Perform logical "inclusive or" bit by bit between the source operand and the destination operand. The result is stored in the destination operand.

```
        Ø OR Ø = Ø
        Ø OR 1 = 1
        1 OR Ø = 1
        1 OR 1 = 1
```

Flag register bits affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
        OR reg,mem/reg

        OR mem/reg,reg

        OR mem/reg,numb
```

Example 1:

```
        OR AX,BX
```

Example 2:

```
        OR CL,41
```

XOR

Description: XOR = exclusive OR

Perform the logical "exclusive or" bit by bit between the source operand and the destination operand. The result is stored in the destination operand.

```
Ø XOR Ø = Ø
Ø XOR 1 = 1
1 XOR Ø = 1
1 XOR 1 = Ø
```

Flag register bit affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
XOR reg,mem/reg

XOR mem/reg,reg

XOR mem/reg,numb
```

Example 1:

```
XOR CL,BL
```

Example 2:

```
XOR AX,Ø1
```

TEST

Description:

Perform the logical "and" of the source operand and the destination operand. The result is not returned to the destination operand, which leaves both operands unchanged. However, it affects flags. When TEST is followed by JNZ (jump if not zero), the jump will be taken if there are "1" bits of the result.

Flag register bits affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
        TEST reg,mem/reg

        TEST mem/reg,reg

        TEST mem/reg,numb
```

Example 1:

```
        TEST BL,34
```

Example 2:

```
        TEST AX,ØFF4
```

RCL, ROL

Description: RCL = rotate through carry left

ROL = rotate left

Rotate the bits in the operand. ROL moves the bits out of the MSB (most significant bit) of the operand and then shift them back to the LSB (least significant bit) of the operand. RCL moves a bit out of the MSB of the operand into the CF. And then shift the CF bit into the empty LSB of the operand. The number of rotation is determined by the count register. If count = 1, source operand is 1; if count > 1, the number of rotation is stored in the CL.

Flag register bits affected: CF, OF.

Syntax:

```
        ROL mem/reg,1

        ROL mem/reg,CL

        RCL mem/reg,1

        RCL mem/reg,CL
```

Example 1:

```
        ROL AX,1
        ROL BYTE[1ØØ],1
```

Example 2:

```
        ROL AX,CL
        ROL BYTE[1ØØ],CL
```

Example 3:

```
        RCL BX,1
        RCL WORD[1ØØ],1
```

Example 4:

```
        RCL BX,CL
        RCL WORD[1ØØ],CL
```

RCR, ROR

Description: RCR = rotate through carry right

ROR = rotate right

ROR moves the bits out of the LSB of the operand and then shift them back to the MSB of the operand. RCR moves a bit out of the LSB of the operand into the CF and then shift the CF bit into the empty MSB of the operand.

Flag register bits affected: CF, OF.

Syntax:

```
        ROR mem/reg,1

        ROR mem/reg,CL

        RCR mem/reg,1

        RCR mem/reg,CL
```

Example 1:

```
        ROR AX,1
```

Example 2:

```
        ROR AX,CL
        ROR BYTE[126],CL
```

Example 3:

```
        RCR BX,1
```

Example 4:

```
        RCR BYTE[127],CL
```

SHL

Description: SHL = shift logical left

This instruction shift the bits in the destination operand to the left. Empty bit positions are filled with zeroes. The number of bits to be shifted is determined by the count register. If count = 1, the source operand is 1; if count > 1, the number of shift is stored in the CL.

Flag register bits affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
        SHL mem/reg,1

        SHL mem/reg,CL
```

Example 1:

```
        SHL BX,1
```

Example 2:

```
        SHL BYTE[126],CL
```

SAR,SHR

Description: SAR = shift arithmetic right

SHR = shift logic right

SAR shifts the bits in the destination operand to the right. The number of shift is determined by the count register. Empty bit positions are filled with the number that equals to the original high-order bit (sign bit) in order that sign of the original operand is retained. SHR shifts the bits in the destination operand to the right. The number of shift is determined by the count register. Empty bit positions are filled with zeroes.

Flag register bits affected: CF, OF, PF, SF, ZF.

AF undefined.

Syntax:

```
        SAR mem/reg,1

        SAR mem/reg,CL

        SHL mem/reg,1

        SHL mem/reg,CL
```

Example 1:

```
        SAR SI,1
```

Example 2:

```
        SAR SI,CL
```

Example 3:

```
        SHR BYTE[123],1
```

Example 4:

```
        SHR BYTE[123],CL
```

4. String-Manipulation Instructions

The 8088 assembly language string-manipulation instructions provide powerful control over strings (bytes or words) of data that are stored in memory.

There are five basic string instructions - MOVS, CMPS, SCAS, LODS and STOS. These instructions are appended with a B or a W (B for a byte, W for a Word, as the case may be) to the mnemonic , so as to indicate whether a byte or a word is to be processed.

The operands for these instructions are implied. The source operand is addressed by the SI (Source Index) register, while the destination operand is addressed by the DI (Destination Index) register. So, when coding these string-manipulation instructions, there is no need to specify the operands.

The source string is always assumed to be in the data segment while the destination string is in the extra segment. The source and destination pointers are updated automatically to point to the next element in the string and this makes it possible for the processor to handle long data strings simply by just repeating the basic string operation a number of times. This process of repeating the operation can be done by prefixing the basic string operation with a repeat code such as REP, REPZ, REPNZ, etc.

When a basic string instruction is prefixed with a repeat code, the processor will repeat the operation of this instruction a number of times equal to the value of the CX register, at the same time subtracting one from the CX register each time the instruction is executed.

For detailed description of these string-manipulation instructions, turn to the following pages.

MOVS

Description: MOVS = Move string byte or word

MOVS instruction moves or transfers a byte or word from the source string to the destination string addressed by the Source Index (SI) and Destination Index (DI) respectively. The source and destination operands can either be registers or memories.

When used together with the prefix REP, MOVS performs memory-to-memory block transfer.

Flag register bits affected: none

Syntax:

1. MOVSB

2. MOVSW

Example :

```
MOV   CX,20 ;Set counter to 20
REP   MOVSW ;Repeat move string word 20 times
```

CMPS

Description: CMPS = Compare string byte or word

Compares the value of the source string (addressed by SI) with the destination string (addressed by DI). When this instruction is executed, a subtraction is performed between the source string and the destination string without actually affecting the contents of either strings. This instruction is used to determine whether the source or the destination string is bigger. But, the statuses of the flag registers are affected after this operation.

The CMPS instruction can be prefixed with JG,JNZ, JZ, REPE, REPZ, REPNE or REPNZ.

Flag register bits affected: AF, CF, OF, PF, SF, ZF

Syntax:

1. CMPSB
2. CMPSW

Examples :

```
MOV   CX,10          ;Set counter to 10
REP   CMPSW          ;Repeat until CX = 0


MOV   CX,5           ;Set Counter to 5
REPNZ CMPSB          ;Repeat compare operation if CX not = 0
                     ;and ZF not = 1
```

SCAS

Description: SCAS = Scan string byte or word

Updates the contents of the flag registers by subtracting the contents of the destination operand (addressed by DI) from the contents of the accumulator (AL if string is a byte or AX if string is a word) register. This operation does not actually alter the contents of the destination operand or the accumulator itself, but merely scans over their contents. The DI is automatically updated after the execution of this instruction.

The SCAS instruction can be prefixed with REPE, REPNE, REPZ OR REPZE.

Flag Register bits affected: AF, CF, OF, PF, SF, ZF

Syntax:

1. SCASB
2. SCASW

Examples:

```
MOV    CX,12          ;Set value of counter.
SCASB                 ;Scan string of OUTPUT1.


MOV   CX,22           ;Set value of counter.
REPNZ SCASW           ;Repeat scanning operation if
                      ;CX not  = Ø and ZF not = 1.
```

LODS

Description: LODS = Load string byte or word

LODS instruction tranfers the contents of the source string operand (addressed by SI) to the accumulator (AL or AX register depending on whether a byte or word is being moved), at the same time updates SI to point to the next element in the string. When prefixed with REP, this instruction will cause the accumulator to be overwritten after each repetition.

Flag register bits affected: none

Syntax:

1. LODSB

2. LODSW

Examples :

```
MOV   CX,11          ;Set value of counter
LODSW                ;Perform word loading operation



MOV   CX,1Ø          ;Set value of counter
REP   LODSB          ;Repeat Loading operation` if CX = Ø
INT   7              ;Else, stop execution.
```

STOS

Description: STOS = Store string byte or word

Stores the contents of the register AX (8 bits, for a byte) or AL (16 bits, for a word) into the memory location addressed by the DI (Destination Index) register and then increments the DI to point to the next location in the string. The STOS instruction can be prefixed with REP.

Flag register bits affected: none

Syntax:

1. STOSB
2. STOSW

Example 1:

```
MOV    CX,18          ;Set value of counter
REP    STOSW          :Repeat store operation until CX = Ø.
```

Example 2:

```
Address       Mnemonics   Operands       Comment
ØØ8Ø:ØØ4Ø      MOV        CX,WORD[45] ;Set value of counter equal
                                      ;to   contents   of   memory
                                      ;location addressed by 45H.
ØØ8Ø:ØØ44      STOSB                  ;perform store operation
ØØ8Ø:ØØ45      DW         12
```

5. Transfer-of-Control Instructions

The transfer-of-control instructions allow the user to transfer control from one point to another in the program. These instructions allow us to alter the sequence of an otherwise straight-line program. The transfer can either be intersegment (from one segment to another) or intrasegment (within one segment).

The transfer-of-control instructions include CALL, RET, JMP, the conditional jump instructions (i.e., JZ, JNZ, etc.), LOOP, the conditional loop instructions (I.e., LOOPE, LOOPNE,etc.), INT and IRET.

CALL

Description: CALL = Call a procedure

The CALL instruction is used to perform a subroutine before returning to the main program that calls the subroutine. When a CALL instruction is encountered, the processor adjusts the IP to point to the next instruction to be executed following the CALL, then saves it on the stack (to allow the RET instruction in the subroutine to return control to the main program), performs the subroutine and finally returns to the main program to continue executing the rest of the instructions.

Flag register bits affected: none

Syntax:

1. CALL procedure
2. CALL dword ptr [addr]
3. CALL reg:off
4. CALL reg

Example :

```
Address    Mnemonics  Operands       Comment
0080:0000  MOV        AX,WORD[102]
0080:0003  CALL       0B            ;Call the routine addressed by
                                    ;value 0B.
0080:0006  POP        AX
0080:0007  MOV        CX,DX
0080:0009  JMP        01A           ;Jumps to the instruction
                                    ;addressed by value 1A.
0080:000B  PUSH       AX
0080:000C  MOV        AX,WORD[FE]
0080:000F  INC        WORD[100]
0080:0013  ADD        AX,WORD[100]
0080:0017  MOV        DX,AX
0080:0019  RET
0080:001A  CMP        AX,CX
```

Description: RET = Return from Procedure

RET returns control from a procedure or subroutine back to the instruction following CALL in the main program. The word at the top of the stack is popped by the RET instruction, then stored in the instruction pointer. The SP (stack pointer) is then incremented by two. If there is an optional pop value, this value is added to the SP. The IP then contains the address of the next instruction following the original CALL instruction in the program.

Flag register bits affected: none

Syntax:

```
        RET

        RET  pop-value

        RETF pop-word
```

Example :

```
Address    Mnemonics  Operands   Comments
0080:0000  CALL       5          ;Invoke routine addressed by memory
                                 ;location 5
0080:0003  JMP        0
0080:0005  MOV        SI,200     ;Move address 200 to SI
0080:0008  CLD
0080:0009  LODSB                 ;Move a byte of data addressed by the SI
                                 ;register into the Al register
0080:000A  CMP        AL,1       ;Check if the end of the predefined data
                                 ;is encountered
0080:000C  JNE        18         ;If data ends, jump to the instruction
                                 ;contained in memory location 18H.
0080:000E  LODSW                 ;Move a word of data addressed by the
                                 ;SI register into AX
0080:000F  MOV        CX,AX      ;Move frequency into CX
0080:0011  LODSW
0080:0012  MOV        BX,AX      ;Move music pitch into BX
0080:0014  INT        18
0080:0016  JMP        9
0080:0018  RET

0080:0200  DB         1
0080:0201  DW         1D5,80
0080:0205  DB         1
0080:0206  DW         1B3,80
0080:020A  DB         1
0080:020B  DW         196,80
```

```
0080:020F   DB          1
0080:0210   DW          184,80
0080:0214   DB          1
0080:0215   DW          16B,80
0080:0219   DB          1
0080:021A   DW          155,80
0080:021E   DB          1
0080:021F   DW          148,80
0080:0223   DB          1
0080:0224   DW          136,80
0080:0228   DB          1
0080:0229   DW          114,80
0080:022D   DB          1
0080:022E   DW          F8,80
0080:0232   DB          1
0080:0233   DW          E6,80
0080:0237   DB          1
0080:0238   DW          B8,80
0080:023C   DB          1
0080:023D   DW          A2,80
0080:0241   DB          1
0080:0242   DW          9A,80
0080:0246   DB          1
0080:0247   DW          88,80
0080:024B   DB          1
0080:024C   DW          78,80
0080:0250   DB          0
```

This program when executed, will produce the basic music notes continuously.

JMP

Description: JMP = Jump

The JMP is an unconditional jump instruction used within a program to transfer control to the target location. The JMP instruction can either be a direct or indirect jump. A jump is direct when the target address is the address contained in the IP (Instruction Pointer), whereas in an indirect jump, the target address is contained in a register or memory address specified in the operand of the JMP instruction.

The JMP instruction can access 65,635 bytes of memory by jumping forward (up to 32,767 bytes) or backward (up to 32,768 bytes). By using the JMP instruction, the user can create a loop for instructions that are repeated a number of times, thereby saving time spent in coding the program and the memory space used to store the program.

Flag register bits affected: none

Syntax:

1. JMP off   (near jump)
2. JMP reg:off (far jump)

Example 1:

```
Address      Mnemonics    Operands     Comments
0080:0000    MOV          DX,1A1       ;Move I/O port 1A1 to DX
0080:0003    MOV          AX,0         ;Set initial value of AX
0080:0006    MOV          CX,0         ;Set initial value of CX
0080:0009    ADD          AX,CX        ;Add CX to AX
0080:000B    ADD          AX,30        ;Obtain ASCII codes from 30H
0080:000E    OUT          DX,AX        ;Output a character  on  the
                                       ;screen
0080:000F    INC          CX
0080:0010    MOV          AX,0
0080:0013    JMP          09           ;Jump to the instruction
                                       ;addressed by location 9
```

Example 2:

```
Address      Mnemonics    Operands     Comments
0080:0000    MOV          DX,16        ;Move target address to Dx
0080:0003    MOV          AX,0         ;Clear accumulator
0080:0006    MOV          CX,5         ;Set initial value of CX
0080:0009    ADD          AX,CX        ;Add contents of CX to
                                       ;accumulator
0080:000B    JMP          DX           ;Get target address from DX
```

Conditional Jump

The conditional jump instructions are used for decision making regarding the program flow if certain conditions are met. To determine whether certain conditions are met or not, the microprocessor tests the contents of some specific flag registers.

Below is a list of the 8Ø88 conditional jumps:

Conditional Jump instructions	Condition: JUMP if	JUMP is performed if:
JA	Above	CF or ZF = Ø
JNBE	Not below or equal	CF or ZF = Ø
JNB	Not below	CF = Ø
JAE	Above or equal	CF = Ø
JB	Below	CF = 1
JNAE	Not above or equal	CF = 1
JC	Carry	CF = 1
JBE	Below or equal	CF or ZF = 1
JNA	Not above	CF or ZF = 1
JE	Equal	ZF = 1
JZ	Zero	ZF = 1
JNLE	Not less or equal	ZF = Ø
JG	Greater	ZF = Ø
JLE	Less or Equal	ZF = 1
JNG	Not greater	ZF = 1
JNL	Not less	SF XOR OF = Ø
JGE	Greater or equal	SF XOR OF = Ø
JL	Lower than	SF XOR OF = 1
JNGE	Note > nor =	SF XOR OF = 1
JNC	Not carry	CF = Ø
JNE	Not equal	ZF = Ø
JNZ	Not zero	ZF = Ø
JNO	Not overflow	OF = Ø
JNP	Not parity	PF = Ø
JPO	Parity odd	PF = Ø
JNS	Positive	SF = Ø
JO	Overflow	OF = 1
JP	Parity	PF = 1
JPE	Parity even	PF = 1
JS	Sign	SF = 1

Flag register bits affected: CF,ZF,SF,PF,OF

Syntax:

```
        J(condition)
```

Example 1:

```
Address      Mnemonics   Operands   Comment
0080:0000    MOV         CX,0       ;Set initial value of CX
0080:0003    MOV         AX,0       ;Clear accumulator
0080:0006    ADD         AX,CX      ;Add CX to AX
0080:0008    ADD         AX,30      ;Obtain ASCII codes from 30H to 80H
0080:000B    INT         B          ;Output a character to the LCD
                                    ;successively
0080:000D    INC         CX         ;Increase CX by 1
0080:000E    CMP         CX,80      ;Check if CX reaches 128 (decimal)
0080:0012    JNE         3          ;Jump to the instruction addressed by
                                    ;the value 3.
0080:0014    INT         7          ;Return to the monitor
```

After the execution of this program, characters corresponding to the ASCII codes 30H to 80H will be displayed on the LCD screen.

Example 2:

```
Address     Mnemonics   Operands   Comments
0080:0000   MOV         BX,10      ;Set BX to 10H (decimal 16)
0080:0003   CMP         BX,F       ;Check if BX is greater than F
0080:0006   JA          A          ;If BX is greater than F, jump to the
                                   ;instruction addressed by the value of A
0080:0008   INT         7          ;If BX is less than F, return to monitor
0080:000A   MOV         AL,31
0080:000C   INT         9          ;Outputs character "1" to the LCD
0080:000E   DEC         AL
0080:0010   INT         9          ;outputs character "0" to the LCD
0080:0012   MOV         AL,3E
0080:0014   INT         9          ;Outputs character ">" to the LCD
0080:0016   MOV         AL,46
0080:0018   INT         9          ;Outputs character "F" to the LCD
0080:001A   MOV         AL,20
0080:001C   INT         9          ;Outputs a blank to the LCD
0080:001E   INT         7          ;Return to the monitor
```

After executing this program, a message "10>F" will be shown on the screen. The purpose of this program is to produce the result of the instruction JA.

LOOP

Unconditional Loop

Description:

LOOP is an unconditional instruction that transfers control to the instruction indicated by the label operand, no matter what the condition of the Flag register. However, the number of times the LOOP instruction is executed depends on the contents of the CX register which serves as a counter for the LOOP. Each time the LOOP instruction is executed, the CX register is decremented by 1 and tested if it is zero. When CX = Ø, the processor stops executing the LOOP instruction and goes on to process the next instruction.

The LOOP instruction sometimes uses a jump instruction called JCXZ (Jump if CX register is Zero) to make its decision on when to start looping and when to get out of the loop. By placing the JCXZ instruction after the instruction that loads the CX register and before the instruction that starts the program loop, the processor tests the content of the CX register first. If, initially it is zero, the LOOP instruction is bypassed, program execution jumps to the next instruction following LOOP. Whereas, in the absence of the JCXZ instruction, upon encountering a LOOP instruction, the first thing the processor does is to subtract the value of CX by 1. At this point, if the initial value of CX is zero, subtracting 1 from CX will give a difference of ØFFFFFH, since ØFFFFFH is now the value of the CX register, the program will have to loop this number of times before it can exit from the loop. To have a better idea on what this is all about, refer to Example 2 below.

Flag register bits affected: none

Syntax:

```
        LOOP address
```

Example 1:

```
Address     Mnemonics   Operands    Comments
ØØ8Ø:ØØØØ   MOV         AX,Ø        ;Set AX to zero
ØØ8Ø:ØØØ3   MOV         CX,[1Ø2]    ;Move the contents of memory location
                                    ;1Ø2 to CX
ØØ8Ø:ØØØ7   CMP         CX,Ø        ;Check if CX equals Ø
ØØ8Ø:ØØØA   JZ          15          ;If CX = Ø, jump to memory address 15
ØØ8Ø:ØØØC   ADD         AX,[1ØØ]    ;Else add the contents of memory
                                    ;location 1ØØ to AX
ØØ8Ø:ØØ1Ø   LOOP        C           ;Perform the instruction addressed by the
                                    ;value  of C a number of times  equal  to
```

```
                                        ;the contents of CX
0080:0012  MOV         [104],AX        ;Store the result into memory address 104
0080:0015  INT         7               ;Return to the monitor
0080:0100  DW          1               ;Initialize location 100 (low byte) to 1
                                        ;and 101 (high byte) to 0.

0080:0102  DW          A               ;Initialize location 102 (low byte) to A
                                        ;and 103 (high byte) to 0
0080:0104  DW          0               ;Clear memory locations 104 and 105 which
                                        ;will be used to store the result of the
                                        ;operation in addition
```

Example 2:

```
Address    Mnemonics   Operands    Comments
0080:0000  MOV         AX,0
0080:0003  MOV         CX,[102]
0080:0007  JCXZ        12          ;Jump if CX = 0
0080:0009  ADD         AX,[100]
0080:000D  LOOP        9           ;Repeat Addition
0080:000F  MOV         [104],AX
0080:0012  INT         7
0080:0100  DW          1
0080:0102  DW          A
0080:0104  DW          0
```

Conditional Loop

Description:

The conditional loop instructions are executed when certain conditions are met. These conditions are reflected by the status of Zero Flag. Keep in mind that the number of times a loop is executed depends on the value of the CX register (this holds true for both conditional and unconditional LOOP), while the Zero Flag only determine whether a loop is to be performed or not.

The conditional loop instructions are as follows:

LOOPE	- Loop while equal
LOOPZ	- Loop while zero
LOOPNE	- Loop while not equal
LOOPNZ	- Loop while not zero

If ZF = 1 and CX register not equal to zero, both LOOPE and LOOPZ will cause the program to loop. In the same manner, if ZF = 0 and CX register not equal to zero, both LOOPNE and LOOPNZ will cause the program to loop.

Flag register bits affected: ZF

Syntax:

1. LOOPE address
2. LOOPZ address
3. LOOPNE address
4. LOOPNZ address

Example 1:

```
Address     Mnemonics   Operands    Comments
0080:0000   ADD         AX,CX
0080:0002   CMP         AX,[10B]    ;Check if AX equals to the contents of
                                    ;memory location 10B
0080:0006   LOOPE       0           ;If equal, repeat addition
0080:0008   MOV         [10B],AX    ;Save contents of AX in 10B
```

Example 2

```
Address     Mnemonics   Operands    Comments
0080:0000   ADD         AX,CX
0080:0002   DEC         CX
0080:0003   LOOPZ       0
0080:0005   INT         7
```

INT

Description: INT = Software Interrupt

The INT instruction is used to initiate a software interrupt, thereby causing a temporary break in the normal execution of a program. Interrupt vectors corresponding to I/O routines were set up in the low memory addresses during initialization. The interrupt vector contains the address of an interrupt service routine.

When an INT instruction is executed, the processor stops whatever it is doing at the moment to service the interrupt, then returns to what it was doing before being interrupted. However, keep in mind that before servicing the interrupt, the processor pushes the contents of the current CS (Code Segment) register into the stack and the high word of the doubleword interrupt pointer is in turn pushed into the CS. Then, the current contents of the IP is pushed into the stack and the contents of the low word of the interrupt pointer is pushed into the IP.

There is a total of 256 interrupt-signal sources. In order to identify the interrupt-signal sources, a interrupt pointer should be specified in the operand field of the INT instruction.

Flag register bits affected: IF,TF

Syntax:

```
        INT     interrupt pointer
```

Example 1:

```
Address     Mnemonics    Operands    Comments
0080:0000   MOV    CX,20             ;Set value of counter
0080:0003   REPNZ MOVSB              ;Repeat move operation if CX not = 0
0080:0005   INT    B                 ;Else INT B
```

IRET

Description: IRET = Return from Interrupt

After servicing an interrupt routine, the processor returns to the program at the point where it was interrupted, through the IRET instruction, which is the final instruction in any interrupt routine. When an IRET instruction is executed, the IP value, CS value and the flag values are popped from the stack and stored in their respective registers. Program execution then continues from the point of interruption.

Flag register bits affected: all

Syntax:

```
        IRET
```

6. Processor-Control Instructions

The processor-control instructions allow the user to set or clear the carry, direction and interrupt flags , invert the current state of the carry flag and even stop instruction executions.

The processor-control instructions consist of the following: CLC, CLD, CLI, CMC, STC, STD and STI.

CLC

Description: Clear Carry Flag

The CLC instruction affects only the carry flag. When a CLC instruction is executed, the carry flag is zeroed out regardless of the state of the carry flag prior to the execution of this instruction.

Flag register bits affected: CF

Syntax:

```
        CLC
```

CLD

Description: CLD = Clear Direction Flag

The CLD instruction only affects the Direction Flag. CLD zeroes out the DF, thereby causing the string instructions to increment the SI and/or DI index registers automatically.

Flag register bits affected: DF

Syntax:

```
        CLD
```

CLI

Description: CLI = Clear Interrupt-Enable Flag

The CLI instruction zeroes out the IF (Interrupt-Enable Flag). When the IF is cleared, maskable interrupts are disabled, that means an external interrupt request that appears on the INTR line will be ignored. However, a non-maskable or a software interrupt is still honored.

Flag register bits affected: IF

Syntax:

```
        CLI
```

CMC

Description: CMC = Complement Carry Flag

The CMC instruction allows us to invert the current state of the carry flag. If the carry flag equals 0, executing the CMC instruction will set it to 1. The CMC instruction only affects the carry flag.

Flag register bits affected: CF

Syntax:

```
        CMC
```

STC

Description: STC = Set Carry

The STC instruction sets the carry flag to 1, regardless of the state of the carry flag prior to the execution of this instruction. Only the carry flag is affected.

Flag register bits affected: CF

Syntax:

```
        STC
```

STD

Description: STD = Set Direction Flag

The STD instruction sets the DF (Direction Flag) to 1 regardless of the state of the Direction Flag prior to the execution of this instruction. STD only affects the DF.

Flag register bits affected: DF

Syntax:

```
        STD
```

STI

Description: STI = Set Interrupt-Enable Flag

STI sets the IF to 1, thereby letting the processor acknowledge maskable interrupt requests on the INTR line after the instruction following STI has been executed.

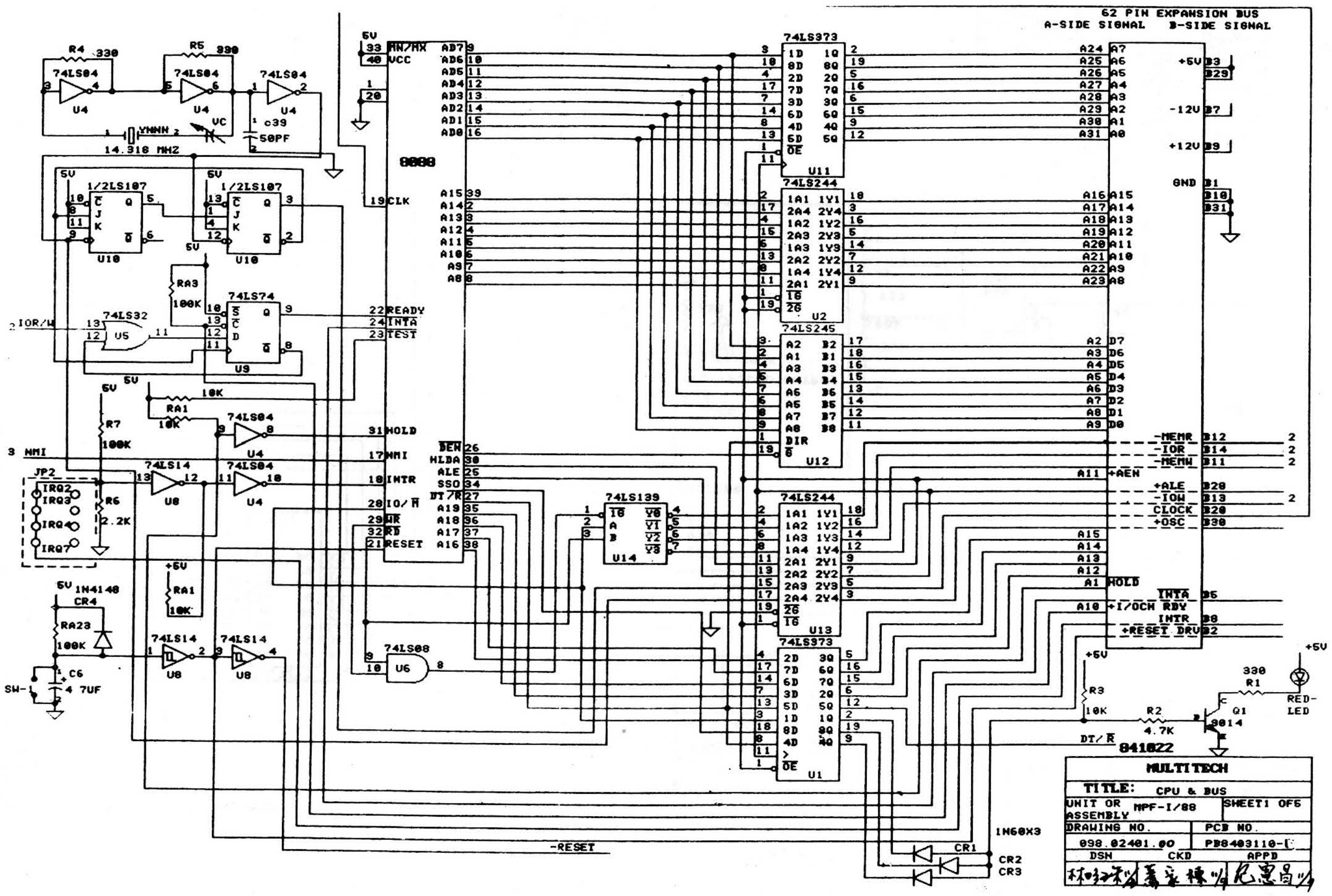
Flag register bits affected: IF

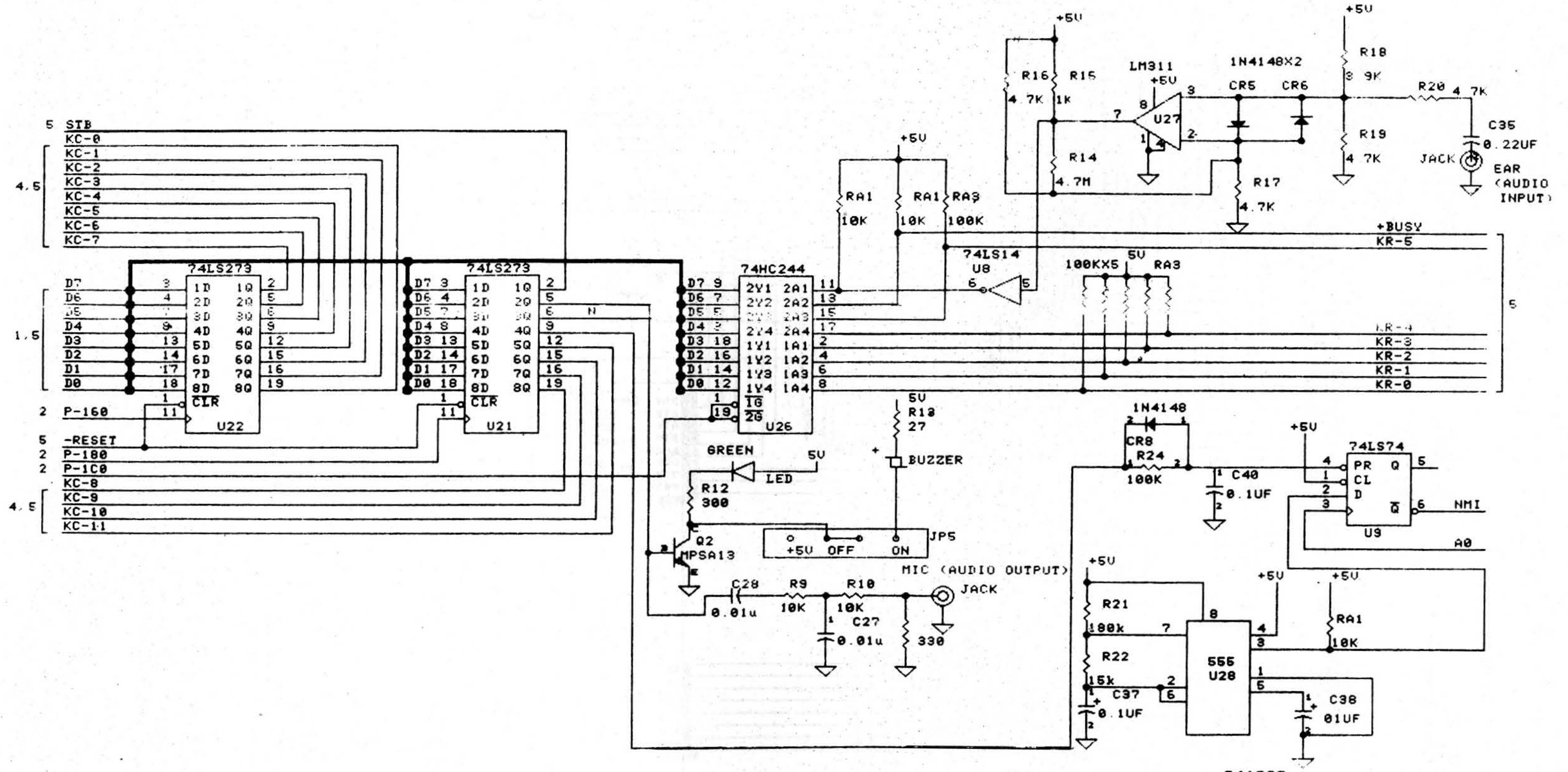
Syntax:

```
        STI
```

Appendix B

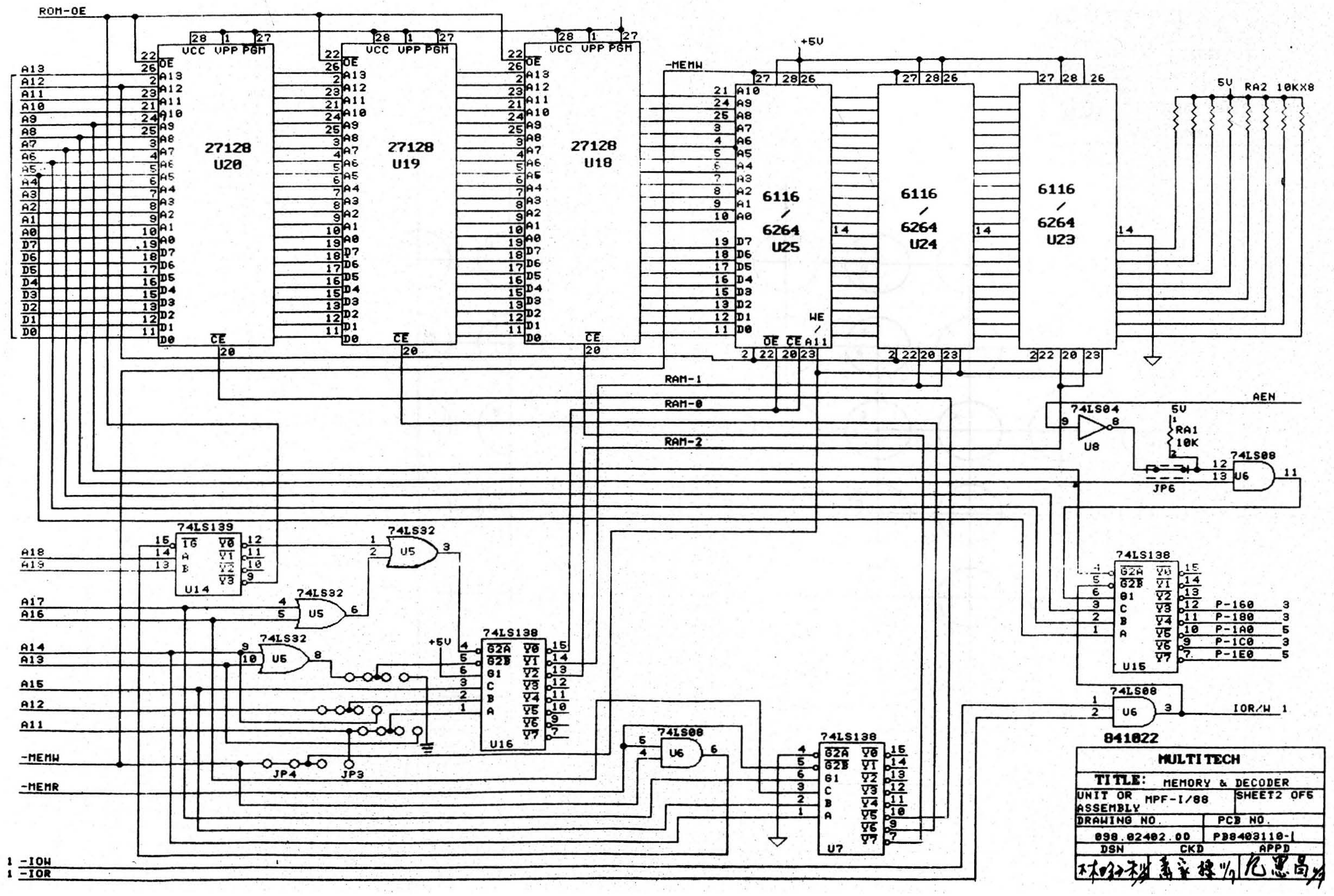
Schematic Diagrams


62 PIN EXPANSION BUS
A-SIDE SIGNAL
B-SIDE SIGNAL
8088
74LS04
74LS107
74LS74
74LS32
74LS14
74LS139
74LS08
74LS373
74LS244
74LS245
U11
U2
U12
U13
U1
U14
14.318 MHZ
JP2
IRQ2
IRQ3
IRQ4
IRQ7
SW-1
1N4148
1N60X3
CR1
CR2
CR3
RED-LED
9014
-RESET
841022
MULTITECH
TITLE: CPU & BUS
UNIT OR ASSEMBLY MPF-I/88
SHEET1 OF5
DRAWING NO. 098.02401.00
PCB NO. PB8403110-[illegible]
DSN
CKD
APPD



841022

MULTITECH		
TITLE: KEYBOARD ,& TAPE		
UNIT OR ASSEMBLY MPF-I/88		SHEET3 OF5
DRAWING NO.		PCB NO
098.02403.00		PB8403110-1
DSN	CKD	APPD
[illegible]	[illegible]	[illegible]



841022

MULTITECH

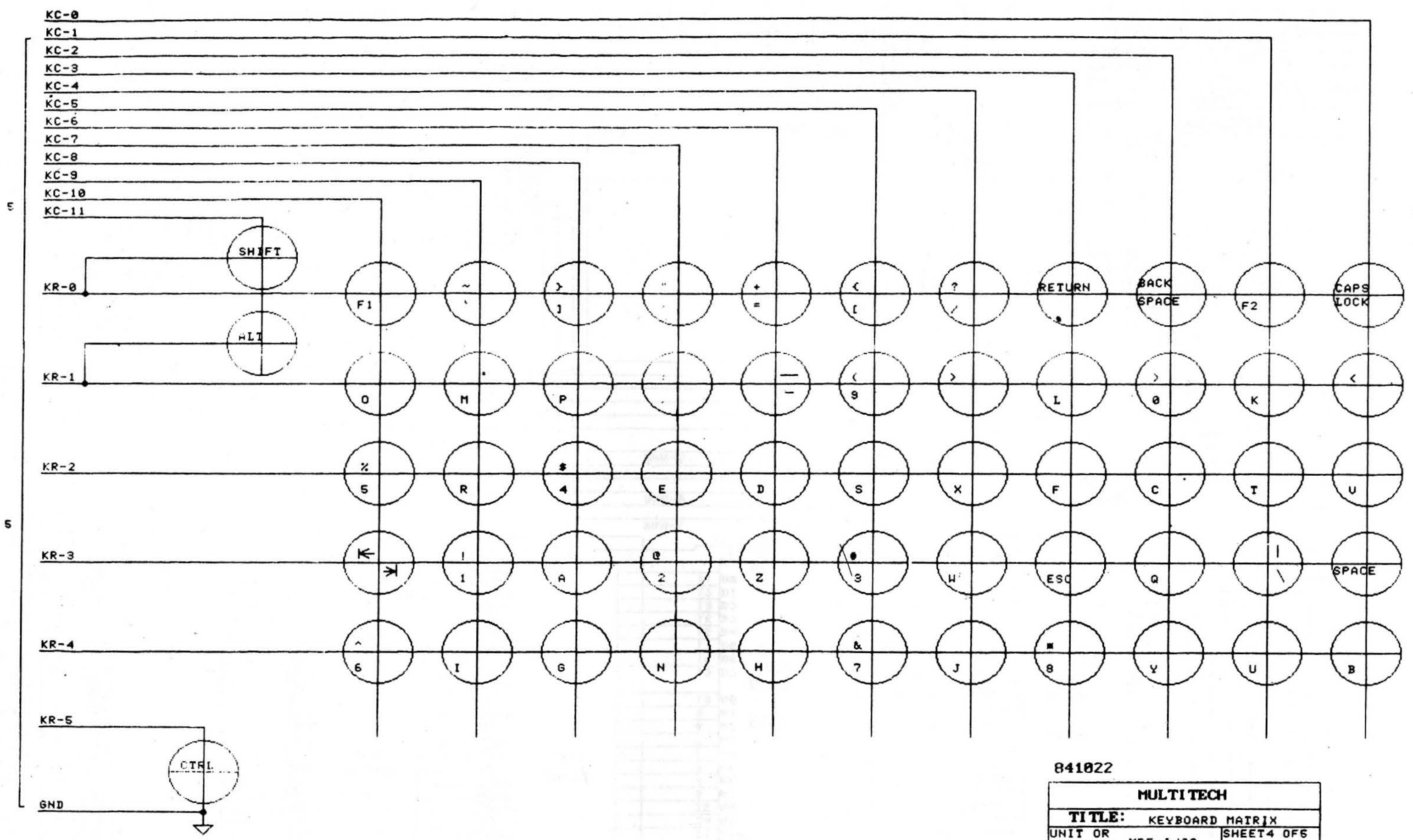
TITLE: MEMORY & DECODER

UNIT OR ASSEMBLY MPF-I/88 | SHEET2 OF6

DRAWING NO. 098.02402.00 | PCB NO. PB8403110-1

DSN	CKD	APPD
[illegible]	[illegible]	[illegible]

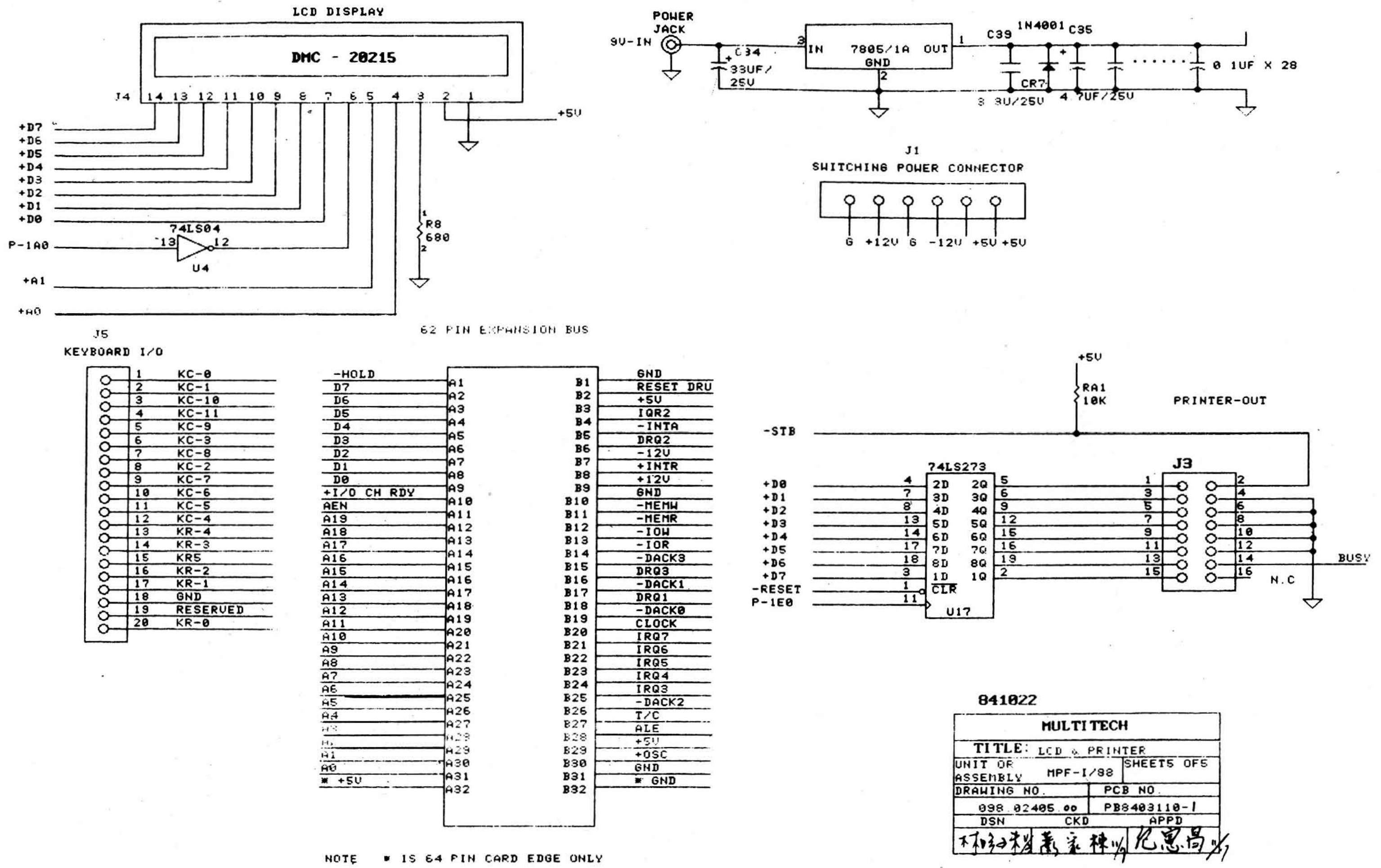
B-3



841022

MULTITECH		
TITLE: KEYBOARD MATRIX		
UNIT OR ASSEMBLY MPF-I/88		SHEET4 OF5
DRAWING NO.	PCB NO.	
098.02404.00	PB8403110-1	
DSN	CKD	APPD
[illegible]	[illegible]	[illegible]

B-4



NOTE * IS 64 PIN CARD EDGE ONLY

841022

MULTITECH	
TITLE: LCD & PRINTER	
UNIT OR ASSEMBLY MPF-I/88	SHEETS OF5
DRAWING NO.	PCB NO.
098 02405 00	PB8403110-1
DSN	CKD	APPD
[illegible]	[illegible]	[illegible]

B-5

Appendix C

Date Sheet of LCD

CONTENTS

1. Outline

The LCD-II (type HD44780) is a dot matrix liquid crystal display controller & driver LSI for displaying alphanumerics, kana characters and symbols. It memorizes character codes (8 bits/character) sent from microcomputers or microprocessors (MPU) into display data RAM (DD RAM, 80 bytes=640 bits, 80 character size), converts them to either 5 x 7 or 5 x 10 dot matrix character patterns, which are then sent to the internal liquid crystal display driver. Since the HD44780 has an internal 16-common signal driver and 40-segment signal driver, one HD44780 can display up to 16 characters (1 character being 5 x 7 dots, 1/16 duty). If a driver LSI HD44100H is externally connected to the HD44780, up to 80 characters can be displayed.

The HD44780 is internally equipped with character generator ROM (CG ROM) that will generate 2 character fonts; 1 font containing 160 5 x 7 dot characters and the other containing 32 5 x 7 dot characters. It is further equipped with character generator RAM (CG RAM, 64 bytes=512 bits) in 8 character size if the character font is 5 x 7 dot, or 4 character size if 5 x 10 dots. CG RAM can be programmed for each application. A feature offering great convenience in actual use. The user can specify any pattern for character-generator ROM. For details, see "The LCD-II (HD44780) Breadboard User's Manual"

To designate character display position, write an instruction into the instruction register from the MPU via data bus and then write a character code into the data register via data bus. Since the HD44780 has a function for automatically shifting the position into which characters are written after character codes by writing only the character code, character displays at serial positions from the next operation are possible. Since the HD44780 also has the function shift the entire display, you can display input from either left or right.

Since both the display data RAM and character generator RAM can be read from the MPU, whatever part not used for display can be used for the general data RAM.

The HD44780 is an 80-pin plastic flat package CMOS LSI. It can transfer data in 4-bit-2-operation or 8-bit-1-operation, allowing either a 4 or 8 bit interface to the MPU. When combined with a CMOS MPU, the user can develop portable battery drive equipment utilizing the liquid crystal display's low power consumption.

2. Features

- 5 x 7 and 5 x 10 dot matrix liquid crystal display controller driver
- Capable of interfacing to 4-bit or 8-bit MPU.
- Display data RAM ....80 x 8 bits (80 characters, max.)
- Character generator ROM....

  Character font 5 x 7 dots: 160 characters

  Character font 5 x 10 dots: 32 characters
- Character generator RAM....

  Programmable; 8 types of 5 x 7 dot character font, or

  4 types of 5 x 10 dot character font
- Both display data and character generator RAMs can be read from the MPU.
- Internal liquid crystal display driver....16 common signal drivers

  40 segment signal drivers
- Duty factor selection (selected by program)....

  1/8 duty: 1 line of 5 x 7 dots + cursor

  1/11 duty: 1 line of 5 x 10 dots + cursor

  1/16 duty: 2 lines of 5 x 7 dots + cursor
- Maximum number of display characters

No. of Display Lines	Duty Factor	Extension	HD44780	HD44100H	No.of Display Characters
1-line display	1/8 1/11 duty	Not provided	1 pc.	----	8 characters x 1 line
		provided	1 pc.	9 pcs.(8 characters/pc.)	80 characters x 1 line
2-line display	1/16 duty	Not provided	1 pc.	----	8 characters x 2 lines
		provided	1 pc	4 pcs. (8 characters x 2 lines/pc)	40 characters x 2 lines

- Wide range of instruction functions

  Display clear, Cursor home, Display ON/OFF, Cursor ON/OFF,

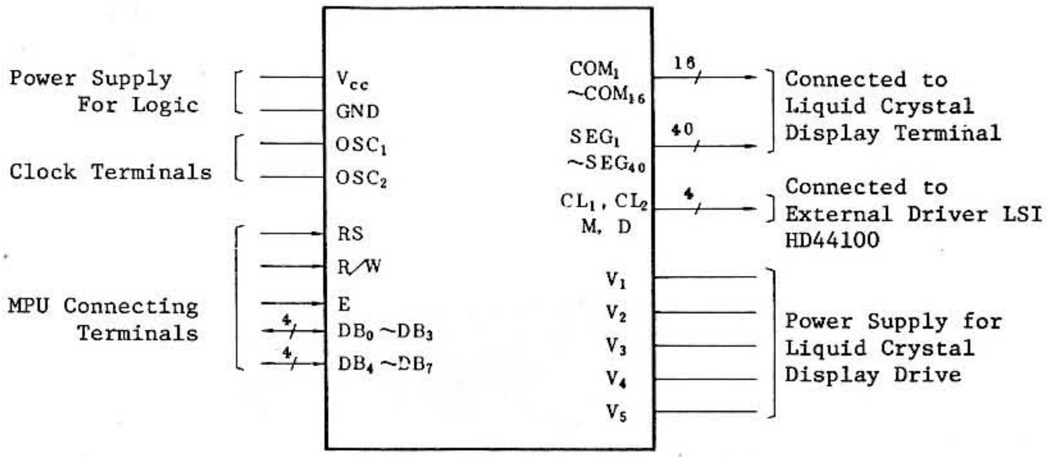
  Display character blink, Cursor shift, Display shift
- Internal automatic reset circuit at power ON. (Internal reset circuit)
- Internal oscillation circuit (with external resistor or ceramic filter)

  (External clock operation possible)
- CMOS process
- Logic power supply; A single+5V (excluding power for liquid crystal display drive)
- Operation temperature range: $-20\sim+75^\circ C$

  (Device for $-40\sim+85^\circ C$ available upon request)
- 80-pin plastic flat package (FP-80)

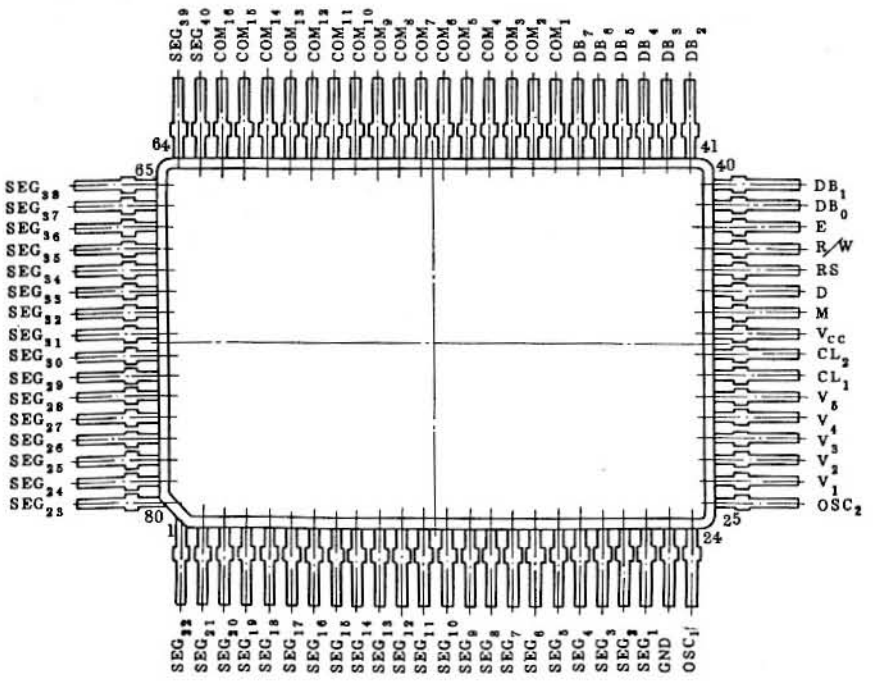
3. Logical Structure and Function

3.1 Symbol Diagram

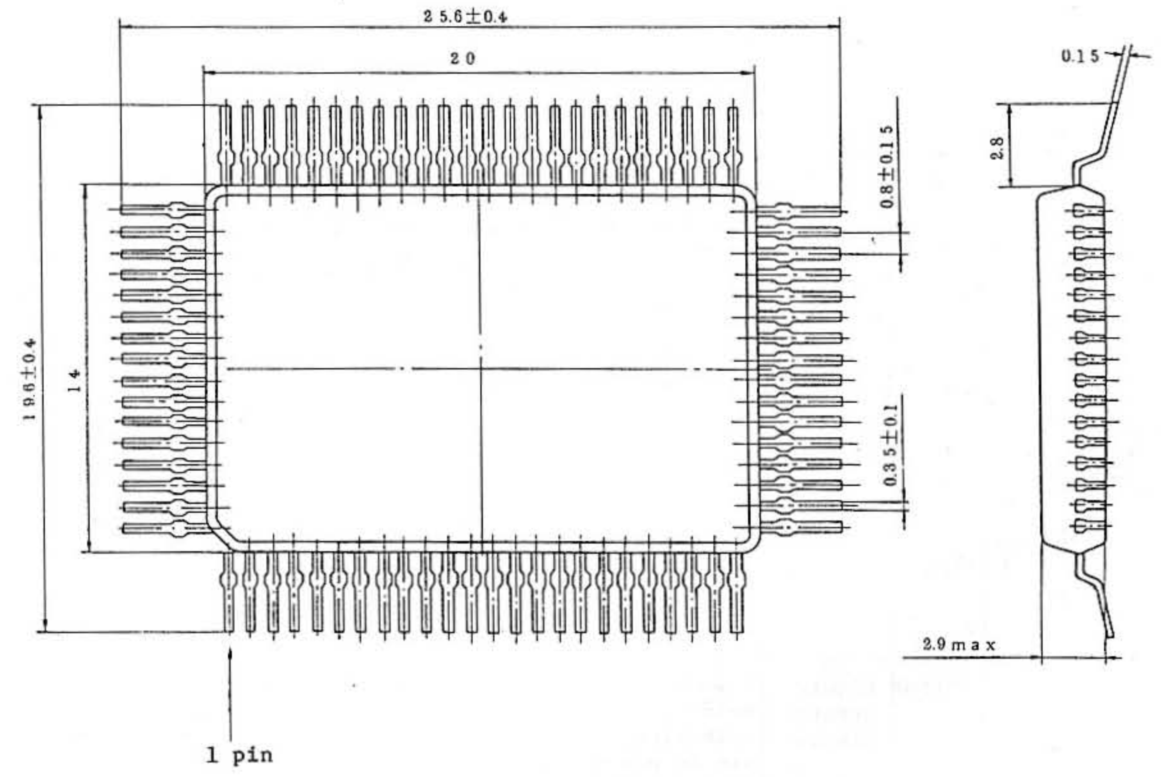


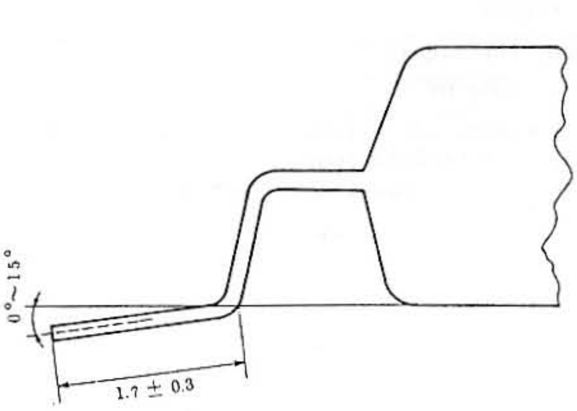
3.2 Pin Assignment and Dimension Outline

(1) Pin Assignment

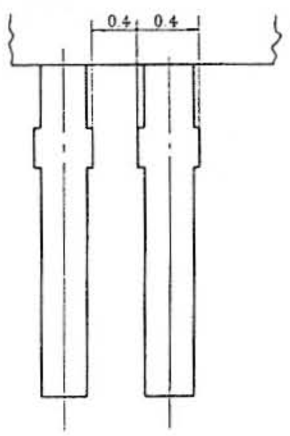


(2) Package Dimension Outline   (80 Pin Plastic Flat Package)





Lead section



Lead section

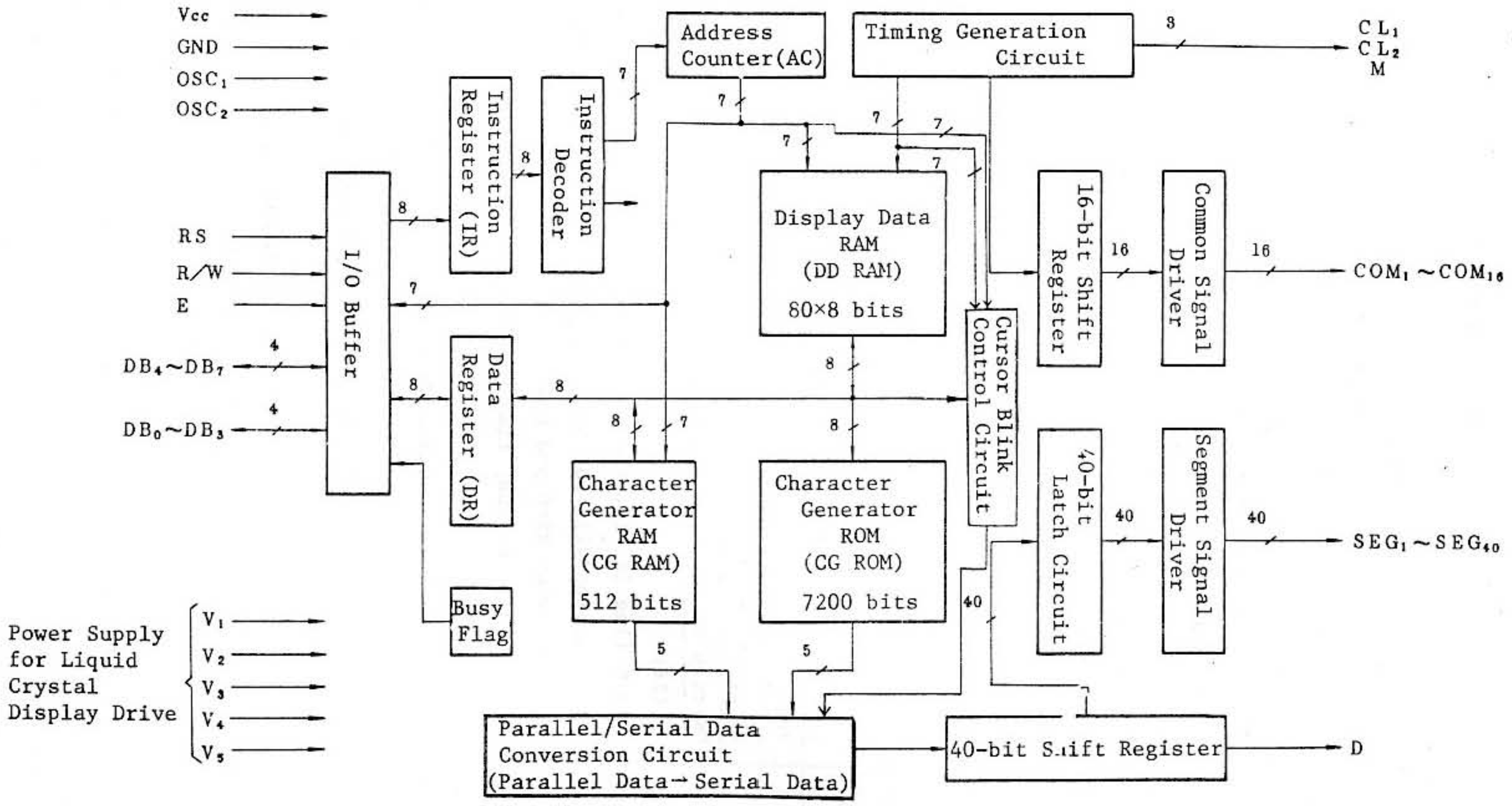
(Unit:mm)

3.3 Terminal Function

Table 3.1 Functional Description of Terminals

Signal name	No. of lines	Input/ Output	Connected to	Function
RS	1	Input	MPU	Signal to select registers "0": Instruction register (for write) Busy flag; address counter (for read) "1": Data register (for read and write)
R/W	1	Input	MPU	Signal to select read (R) and write (W) "0": Write "1": Read
E	1	Input	MPU	Operation start signal for data read / write
$DB_4 \sim DB_7$	4	Input/ Output	MPU	Higher order 4 lines data bus with bidirectional tri-state. Used for data transfer between the MPU and the HD44780. DB7 can be used as a BUSY flag.
$DB_0 \sim DB_3$	4	Input/ Output	MPU	Lower order 4 lines data bus with bidirectional tri-state. Used for data transfer between the MPU and the HD44780. These four are not used during 4-bit operation.
CL_1	1	Output	HD44100H	Clock to latch serial data D sent to the driver LSI HD44100.
CL_2	1	Output	HD44100H	Clock to shift serial data D.
M	1	Output	HD44100H	Switch signal to convert liquid crystal drive waveform to AC
D	1	Output	HD44100H	Character pattern data corresponding to each common signal is serially sent. "0": Non selection "1": Selection
$COM_1 \sim COM_{16}$	16	Output	Liquid crystal display	Common signals that are not used are charged to non-selection waveforms. That is, $COM_9 \sim COM_{16}$ are in non-selection waveform at 1/8 duty factor, and $COM_{12} \sim COM_{16}$ are in non-selection waveform at 1/11 duty factor.
$SEG_1 \sim SEG_{40}$	40	Output	Liquid crystal display	Segment signal
$V_1 \sim V_5$	5		Power supply	Power supply for liquid crystal display drive
V_{cc}, GND	2		Power supply	V_{cc}; +5V, GND; 0V
OSC_1, OSC_2	2			Terminals connected to resistor or ceramic filter for internal clock oscillation. For external clock operation, the clock is input to OSC_1.

3.4 Block Diagram (HD44780 Interior)



3.5 Function of Each Block

(1) Register

The HD44780 has two 8-bit registers, an instruction register (IR) and a data register (DR).
The IR stores instruction codes such as display clear and cursor shift, and address information for display data RAM (DD RAM) and character generator RAM (CG RAM). The IR can be written from the MPU but not read by the MPU.
The DR temporarily stores data to be written into the DD RAM or the CG RAM and data to be read out from DD RAM or CG RAM. Data written into the DR from the MPU is automatically written into the DD RAM or the CG RAM by internal operation. The DR is also used for data storage when reading data from the DD RAM or the CG RAM. When address information is written into the IR, data is read into the DR from the DD RAM or the CG RAM by internal operation. Data transfer to the MPU is then completed by the MPU reading DR. After the MPU reads the DR, data in the DD RAM or CG RAM at the next address is sent to the DR for the next read from the MPU. Register selector (RS) signals make their selection from these two registers.

Table 3.2 Register selection

RS	R/W	Operation
0	0	IR write as internal operation (Display clear, etc.)
0	1	Read busy flag (DB_7) and address counter (DB_0~DB_6)
1	0	DR write as internal operation (DR to DD or CG RAM)
1	1	DR read as internal operation (DD or CG RAM to DR)

(2) Busy flag (BF)

When the busy flag is "1", the HD44780 is in the internal operation mode, and the next instruction will not be accepted. As Table 3.2 shows, the busy flag is output to DB_7 when RS=0 and R/W=1. The next instruction must be written after ensuring that the busy flag is "0".

(3) Address counter (AC)

The address counter (AC) assigns addresses to DD and CG RAMs. When an instruction for address is written in IR, the address information is sent from IR to AC. Selection of either DD or CG RAM is also determined concurrently by the instruction.
After writing into (or reading from) DD or CG RAM display data, AC is automatically incremented by +1 (or decremented by -1). AC contents are output to DB_0~DB_6 when RS=0 and R/W=1, as shown in Table 3.2.

(4) Display data RAM (DD RAM)

The display data RAM (DD RAM) stores display data represented in 8-bit character codes. Its capacity is 80×8 bits, or 80 characters. The display data RAM (DD RAM) that is not used for display can be used as a general data RAM. Relations between DD RAM addresses and positions on the liquid crystal display are shown below.

The DD RAM address (A_{DD}) is set in the Address Counter (AC) and is represented in hexadecimal.

← Upper Order Bits   Lower Order Bits →

AC	AC 6	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0

Hexadecimal

(Example) DD RAM address "4E"

1	0	0	1	1	1	0
4			E			

•1-line Display (N=0)

(digit)	1	2	3	4	5		79	80 ← Display Position
1-line	0 0	0 1	0 2	0 3	0 4	....	4 E	4 F ← DD RAM Address

(a) When the display characters are less than 80, the display begins at the head position. For example, 8 characters using 1 HD44780 are displayed as :

(digit)	1	2	3	4	5	6	7	8 ← Display Position
1-line	0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7 ← DD RAM Address

When the display shift operation is performed, the DD RAM address moves as:

(Left Shift Display)	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8
(Right Shift Display)	4 F	0 0	0 1	0 2	0 3	0 4	0 5	0 6

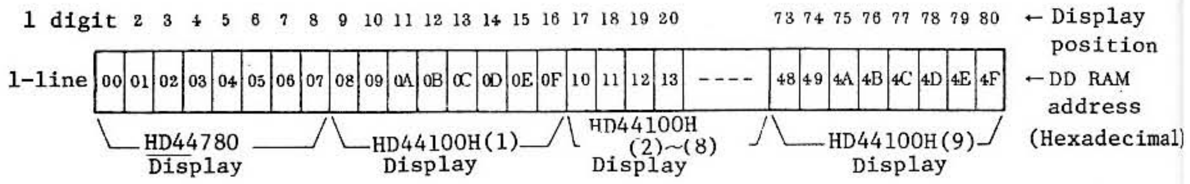
(b) 16-character display using an HD44780 and an HD44100H is as shown below:

(digit)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16 ← Display Position
1-line	0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	0 A	0 B	0 C	0 D	0 E	0 F ← DD RAM Address
	HD44780 Display								HD44100H Display							

When the display shift operation is performed, the DD RAM address moves as:

(Left Shift Display)	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	0 A	0 B	0 C	0 D	0 E	0 F	1 0
(Right Shift Display)	4 F	0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	0 A	0 B	0 C	0 D	0 E

(c) The relation between display position and DD RAM address when the number of display digits is increased through the use of one HD44780 and two or more HD44100H's can be considered an extension of (b).
Since the increase can be 8 digits for each additional HD44100H, up to 80 digits can be displayed by externally connecting 9 HD44100H's.


1 digit 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 73 74 75 76 77 78 79 80
← Display position
1-line
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 ---- 48 49 4A 4B 4C 4D 4E 4F
← DD RAM address (Hexadecimal)
HD44780 Display
HD44100H(1) Display
HD44100H (2)~(8) Display
HD44100H(9) Display

•2-line Display (N=1)

(digit)	1	2	3	4	5		39	40
1-line	0 0	0 1	0 2	0 3	0 4	....	2 6	2 7
2-line	4 0	4 1	4 2	4 3	4 4	....	6 6	6 7

← Display Position
← DD RAM Address

(a) When the number of display characters is less than 40 x 2 lines, the 2 lines from the head are displayed. Note that the first line end address and the second line start address are not consecutive. For example when an HD44780 is used, 8 characters × 2 lines are displayed as:

(digit)	1	2	3	4	5	6	7	8
1-line	0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7
2-line	4 0	4 1	4 2	4 3	4 4	4 5	4 6	4 7

← Display Position
← DD RAM Address

When display shift is performed, the DD RAM address moves as:

(Left Shift Display)	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8
	4 1	4 2	4 3	4 4	4 5	4 6	4 7	4 8

(Right Shift Display)	2 7	0 0	0 1	0 2	0 3	0 4	0 5	0 6
	6 7	4 0	4 1	4 2	4 3	4 4	4 5	4 6

(b) 16 characters × 2 lines are displayed when an HD44780 and an HD44100H are used:

(digit)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	← Display Position
1-line	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	← DD RAM Address
2-line	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	

HD44780 Display (digits 1–8) / HD44100H Display (digits 9–16)

When display shift is performed, the DD RAM address moves as follows:

(Left Shift Display)																
	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50

(Right Shift Display)																
	27	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
	67	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E

(c) The relation between display position and DD RAM address when the number of display digits is increased by using one HD44780 and two or more HD44100H's, can be considered an extension of (b).
Since the increase can be 8 ditits x 2 lines for each additional HD44100H, up to 40 digits x 2 lines can be displayed by connecting 4 HD44780's externallv.

1 digit	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		33	34	35	36	37	38	39	40	← Display position
1-line 00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	-----	20	21	22	23	24	25	26	27	← DD RAM address (Hexadecimal)
2-line 40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	-----	60	61	62	63	64	65	66	67	

HD44780 Display / HD44100H(1) Display / HD44100H (2)(3) Display / HD44100H(4) Display

(5) Character Generator ROM (CG ROM)

The character generator ROM generates 5 x 7 dot or 5 x 10 dot character patterns from 8-bit character codes. It can generate 160 types of 5 x 7 dot character patterns and 32 types of 5 x 10 dot character patterns. Table 3.3 and 3.4 show the relation between character codes and character patterns in the Hitachi standard HD44780A00. User defined character patterns are also available by mask-programming ROM. For details, see "The LCD-II(HD44780) Breadboard User's Manual".

(6) Character Generator RAM (CG RAM)

The character generator RAM is the RAM with which the user can rewrite character patterns by program. With 5 x 7 dots, 8 types of character patterns can be written and with 5 x 10 dots 4 types can be written. Write the character codes in the left columns of Tables 3.3 and 3.4 to display character patterns stored in CG RAM.

Table 3.5 shows the relation between CG RAM addresses and data and display patterns.

As Table 3.5 shows, an area that is not used for display can be used as a general data RAM.

Table 3.3 Correspondence between Character Codes and Character Pattern
(Hitachi standard HD44780A00)

Higher 4bit / Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)		0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101	(6)	-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Table 3.4 Correspondence between Character Codes and Characte
(Hitachi standard HD44780A00)

Higher 4bit / Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)		Ø	@	P	\	p		ー	タ	ミ	α	p
xxxx0001	(2)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010	(3)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011	(4)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100	(5)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110	(7)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	$\bar{x}$
xxxx1001	(2)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	$^{-1}$	y
xxxx1010	(3)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011	(4)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	x	万
xxxx1100	(5)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101	(6)	−	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)	.	>	N	^	n	→	ョ	セ	ホ	゛	$\bar{n}$	
xxxx1111	(8)	/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Table 3.5 Relation between CG RAM Addresses and Character Codes (DD RAM) and Character Patterns (CG RAM Data)

(a) For 5×7 dot character patterns

Character Codes (DD RAM Data)	CG RAM Address	Character Patterns (CG RAM Data)	
7 6 5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→	5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→	7 6 5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→	
	0 0 0	* * * 1 1 1 1 0	
	0 0 1	1 0 0 0 1	
	0 1 0	1 0 0 0 1	Character Pattern Example (1)
	0 1 1	1 1 1 1 0	
0 0 0 0 * 0 0 0	0 0 0 1 0 0	1 0 1 0 0	
	1 0 1	1 0 0 1 0	
	1 1 0	1 0 0 0 1	
	1 1 1	* * * 0 0 0 0 0	← Cursor Position
	0 0 0	* * * 1 0 0 0 1	
	0 0 1	0 1 0 1 0	
	0 1 0	1 1 1 1 1	Character Pattern Example (2)
	0 1 1	0 0 1 0 0	
0 0 0 0 * 0 0 1	0 0 1 1 0 0	1 1 1 1 1	
	1 0 1	0 0 1 0 0	
	1 1 0	0 0 1 0 0	
	1 1 1	* * * 0 0 0 0 0	
	0 0 0	* * *	
	0 0 1		
0 0 0 0 * 1 1 1	1 1 1		
	1 0 0		
	1 0 1		* No effect
	1 1 0		
	1 1 1	* * *	

(Note) 1: Character code bits 0∿2 correspond to CG RAM address bits 3∿5 (3 bits:8 types).

2: CG RAM address bits 0∿2 designate character pattern line position. The 8th line is the cursor position and display is performed in logical OR by the cursor.
Maintain the 8th line data, corresponding to the cursor display position, in the "0" state for cursor display. When the 8th line data is "1", bit 1 lights up regardless of cursor existence.

3: Character pattern row positions correspond to CG RAM data bits 0 4, as shown in the figure (bit 4 being at the left end). Since CG RAM data bits 5∿7 are not used for display, they can be used for the general data RAM.

4: As shown in Table 3.3 and 3.4, CG RAM character patterns are selected when character code bits 4∿7 are all "0". However, since character code bit 3 is a ineffective bit, the "R" display in the character pattern example, is selected by character code "00" (hexadecimal) or "08" (hexadecimal).

5: "1" for CG RAM data corresponds to selection for display and "0" for non-selection.

(b) For 5×10 dot character patterns

Character Codes (DD RAM Data)	CG RAM Address	Character Patterns (CG RAM Data)
7 6 5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→	5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→	7 6 5 4 3 2 1 0 ←Higher Order Bits Lower Order Bits→
	0 0 0 0	* * * 0 0 0 0 0
	0 0 0 1	↓ 0 0 0 0 0
	0 0 1 0	1 0 1 1 0
	0 0 1 1	1 1 0 0 1
	0 1 0 0	1 0 0 0 1
0 0 0 0 * 0 0 *	0 0 0 1 0 1	1 0 0 0 1
	0 1 1 0	1 1 1 1 0
	0 1 1 1	1 0 0 0 0
	1 0 0 0	1 0 0 0 0
	1 0 0 1	1 0 0 0 0
	1 0 1 0	* * * 0 0 0 0 0
	1 0 1 1	* * * * * * * *
	1 1 0 0	↓ ↓
	1 1 0 1	
	1 1 1 0	
	1 1 1 1	* * * * * * * *
	0 0 0 0	* * *
	0 0 0 1	↓
0 0 0 0 * 1 1 *	1 1 1 0 0 1	
	1 0 1 0	* * *
	1 0 1 1	* * * * * * * *
	1 1 0 0	↓ ↓
	1 1 0 1	
	1 1 1 0	
	1 1 1 1	* * * * * * * *

Character Pattern Example

Cursor ←Position

* No Effect

(Note) 1: Character code bits 1, 2 correspond to CG RAM address bits 4,5 (2 bits:4 types).

2: CG RAM address bits 0~3 designate character pattern line position. The 11th line is the cursor position and display is performed in logical OR with cursor.
Maintain the 11th line data corresponding to the cursor display position in the "0" state for cursor display. When the 11th line data is "1", bit 1 lights up regardless of cursor existence. Since the 12th~16th lines are not used for display, they can be used for the general data RAM.

3: Character pattern row positions are the same as 5 x 7 dot character pattern positions.

4: CG RAM character patterns are selected when character code bits 4~7 are all "0". However, since character code bit 0 and 3 are ineffective bits, "P" display in the character pattern example is selected by e character code "00", "01", "08" and "09" (hexadecimal).

5: "1" for CG RAM data corresponds to selection for display and "0" for non-selection.

(7) Timing Generation Circuit

The timing generation circuit generates timing signals to operate internal circuits such as DD RAM, CG ROM and CG RAM. RAM read timing needed for display and internal operation timing by MPU access are separately generated so they do not interfere with each other. Therefore, when writing data to the DD RAM,for example, there will be no undesirable influence, such as flickering, in areas other than the display area. This circuit also generates timing signals to operate the externally connected driver LSI HD44100H.

(8) Liquid Crystal Display Driver Circuit

The liquid crystal display driver circuit consists of 16 common singal drivers and 40 segment signal drivers. When character font and number of lines are selected by a program, the required common signal drivers automatically output drive waveforms, the other common signal drivers continue to output non-selection waveforms.

The segment signal driver has essentially the same configuration as the driver LSI HD44100H (see Fig. 6.12). Character pattern data is sent serially through a 40-bit shift register and latched when all needed data has arrived. The latched data controls the driver for generating drive waveform outputs.

The serial data is sent to the HD44100H, externally connected in cascade, used for display digit number extension.

Send of serial data always starts at the display data character pattern corresponding to the last address of the display data RAM (DD RAM).

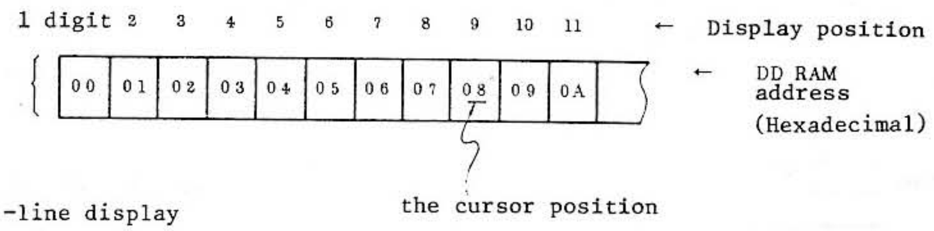
Since serial data is latched when the display data character pattern, corresponding to the starting address, enters the internal shift register, the HD44780 drives the head display. The rest displays, corresponding to latter addresses, are added with each additional HD44100H.

(9) Cursor / Blink Control Circuit

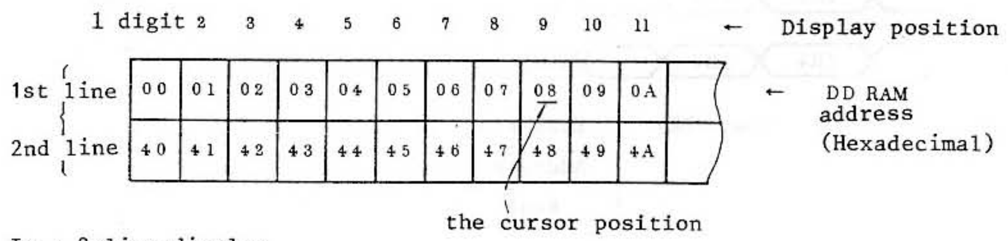
This is the circuit that generates the cursor or blink. The cursor or the blink appear in the digit residing at the display data RAM (DD RAM) address set in the address counter (AC).

When the address counter is (08) 16, a cursor position is :

	AC6	AC5	AC4	AC3	AC2	AC1	AC0
AC	0	0	0	1	0	0	0


1 digit 2 3 4 5 6 7 8 9 10 11
← Display position
00 01 02 03 04 05 06 07 08 09 0A
← DD RAM address (Hexadecimal)
the cursor position

In a 1-line display


1 digit 2 3 4 5 6 7 8 9 10 11
← Display position
1st line
00 01 02 03 04 05 06 07 08 09 0A
2nd line
40 41 42 43 44 45 46 47 48 49 4A
← DD RAM address (Hexadecimal)
the cursor position

In a 2-line display

(Note) The cursor or blink appears when the address counter (AC) selects the character generator RAM (CG RAM). But the cursor and blink are meaningless.

The cursor or blink is displayed in the meaningless position when AC is the CG RAM address.

3.6 Interfacing to MPU

In the HD44780, data can be sent in either 4-bit 2-operation or 8-bit 1-operation so it can interface to both 4 and 8 bit MPU's.

(1) When interface data is 4-bits long, data is transferred using only 4 buses : $DB_4 \sim DB_7$. $DB_0 \sim DB_3$ are not used. Data transfer between the HD44780 and the MPU completes when 4-bit data is transferred twice. Data of the higher order 4 bits (contents of $DB_4 \sim DB_7$ when interface data is 8 bits long) is transferred first, then the lower order 4 bits (content of $DB_0 \sim DB_3$ when interface data is 8 bits long) is transferred. Check the busy flag after 4-bit data has been transferred twice (one instruction). A 4-bit 2-operation will then transfer the busy flag and address counter data.

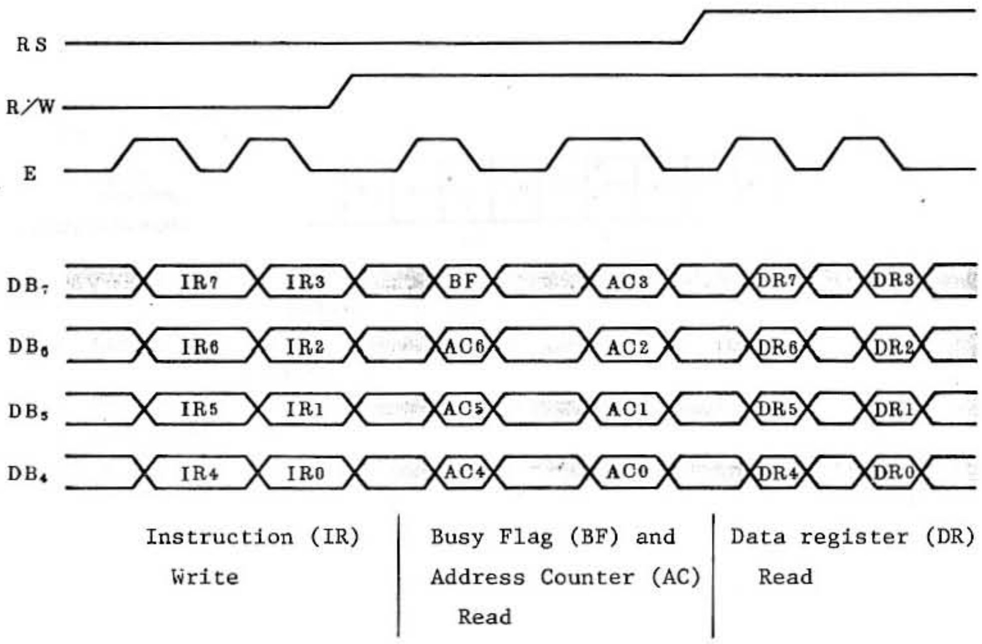


Fig. 3.1 4-bit Data Transfer Example

(2) When interface data is 8 bits long, data is transferred using the 8 data buses of $DB_0 \sim DB_7$.

3.7 Reset Function

3.7.1 Initializing by Internal Reset Circuit

The HD44780 automatically initializes (resets) when power is turned on using the internal reset circuit. The following instructions are executed in initialization. The busy flag (BF) is kept in busy state until initialization ends. (BF=1) The busy state is 10 ms after Vcc rises to 4.5V.

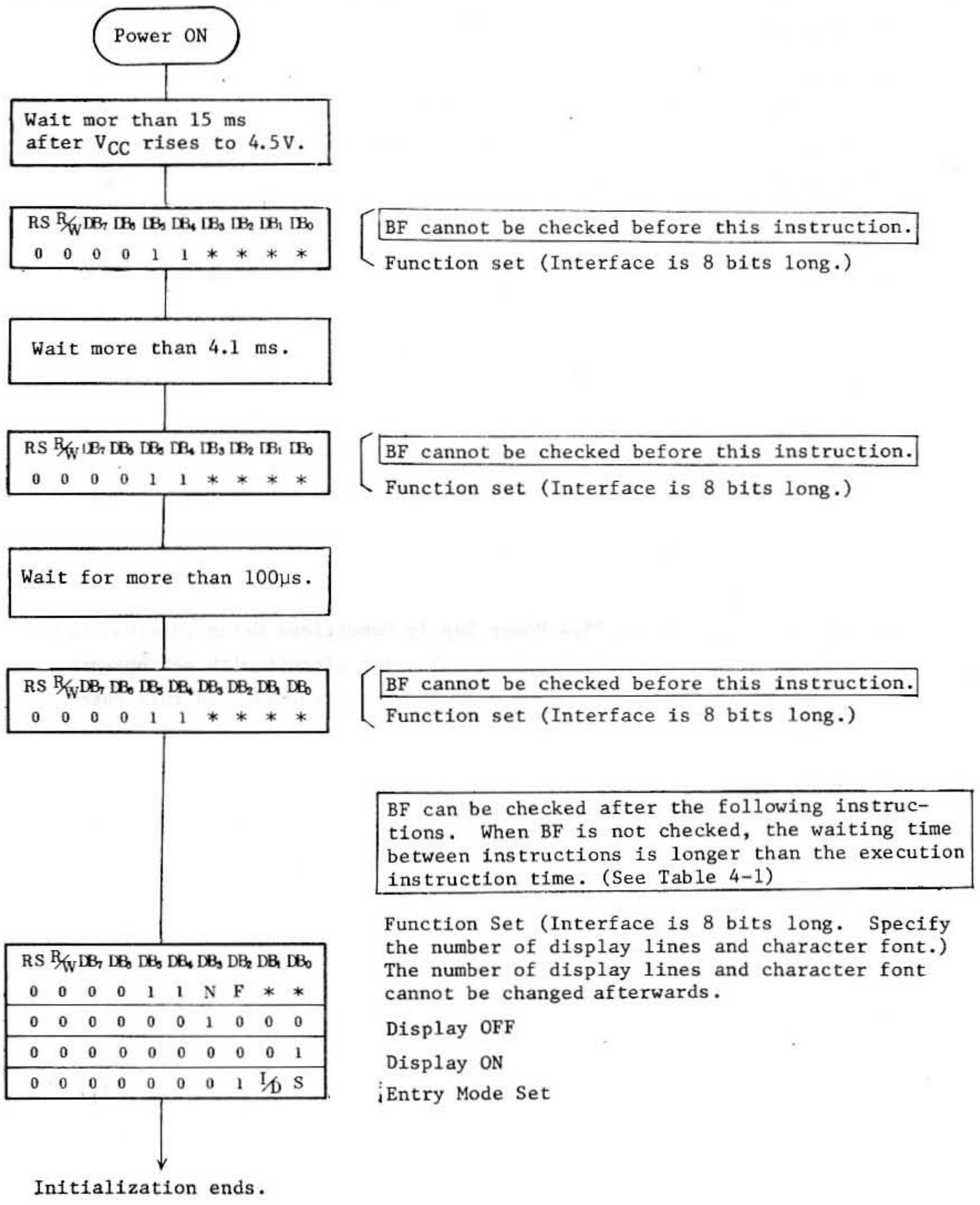
(1) Display clear		
(2) Function ser	DL=1	: 8 bit long interface data
	N =0	: 1-line display
	F =0	: 5 x 7 dot character font
(3) Display ON/OFF control	D =0	: Display OFF
	C =0	: Cursor OFF
	B =0	: Blink OFF
(4) Entry mode set	I/D=1	: +1 (increment)
	S =0	: No shift

(Note) When conditions in "5.4 Power Supply Conditions Using Internal Reset Circuit" are not met, the internal reset circuit with not operate normally and initialization will not be performed. In this case initialize by MPU according to "3.7.2 Initializing by Instruction".

3.7.2 Initializing by Instruction

If the power supply conditions for correctly operating the internal reset circuit are not met, initialization by instruction is required.
Use the following procedure for initialization.

(1) When interface is 8 bits long ;



Power ON

Wait mor than 15 ms after V_{CC} rises to 4.5V.

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.

Function set (Interface is 8 bits long.)

Wait more than 4.1 ms.

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.

Function set (Interface is 8 bits long.)

Wait for more than 100μs.

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	1	1	*	*	*	*

BF cannot be checked before this instruction.

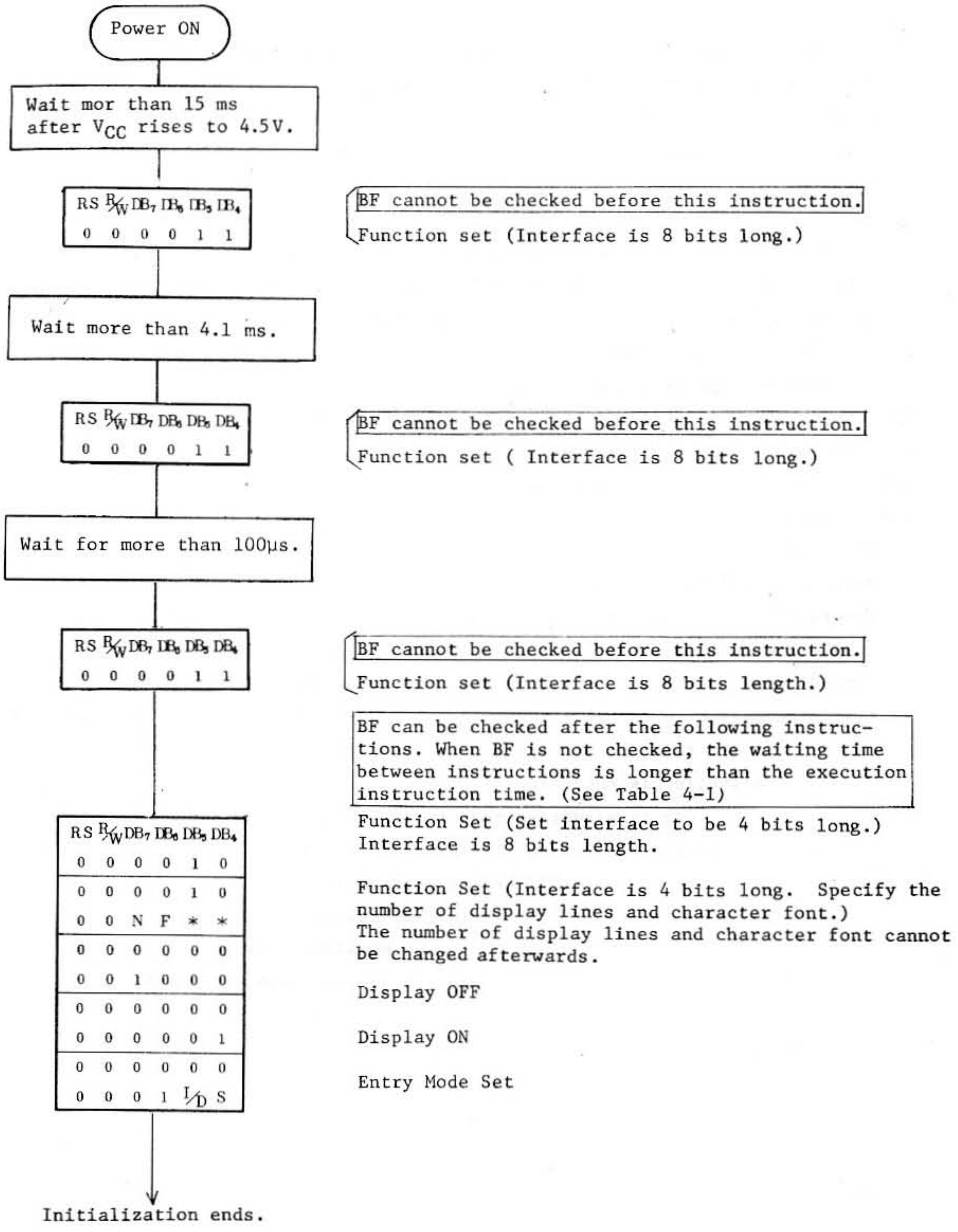
Function set (Interface is 8 bits long.)

BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instruction time. (See Table 4-1)

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	
0	0	0	0	1	1	N	F	*	*	Function Set (Interface is 8 bits long. Specify the number of display lines and character font.) The number of display lines and character font cannot be changed afterwards.
0	0	0	0	0	0	1	0	0	0	Display OFF
0	0	0	0	0	0	0	0	0	1	Display ON
0	0	0	0	0	0	0	1	I/D	S	Entry Mode Set

Initialization ends.

(2) When interface is 4 bits long ;



Power ON

Wait mor than 15 ms after V_{CC} rises to 4.5V.

RS	R/W	DB_7	DB_6	DB_5	DB_4
0	0	0	0	1	1

BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)

Wait more than 4.1 ms.

RS	R/W	DB_7	DB_6	DB_5	DB_4
0	0	0	0	1	1

BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)

Wait for more than 100μs.

RS	R/W	DB_7	DB_6	DB_5	DB_4
0	0	0	0	1	1

BF cannot be checked before this instruction.
Function set (Interface is 8 bits length.)

BF can be checked after the following instructions. When BF is not checked, the waiting time between instructions is longer than the execution instruction time. (See Table 4-1)

RS	R/W	DB_7	DB_6	DB_5	DB_4	
0	0	0	0	1	0	Function Set (Set interface to be 4 bits long.) Interface is 8 bits length.
0	0	0	0	1	0	Function Set (Interface is 4 bits long. Specify the number of display lines and character font.) The number of display lines and character font cannot be changed afterwards.
0	0	N	F	*	*	
0	0	0	0	0	0	Display OFF
0	0	1	0	0	0	
0	0	0	0	0	0	Display ON
0	0	0	0	0	1	
0	0	0	0	0	0	Entry Mode Set
0	0	0	1	I/D	S	

Initialization ends.

4. Instruction

4.1 Outline

Only two HD44780 registers, the Instruction Register (IR) and the Data Register (DR) can be directly controlled by the MPU. Prior to internal operation start, control information is temporarily stored in these registers, to allow interface from HD44780 internal operation to various types of MPUs which operate in different speeds or to allow interface to peripheral control ICs. HD44780 internal operation is determined by signals sent from the MPU. These signals include register selection signals (RS), read/write signals (R/W) and data bus signals ($DB_0 \sim DB_7$), and are called instructions, here. Table 4.1 shows the instructions and their execution time. Details are explained in subsequent sections. Instructions are of 4 types, those that,

(1) Designate HD44780 functions such as display format, data length, etc.

(2) Give internal RAM addresses.

(3) Perform data transfer with internal RAM

(4) Others

In normal use, category (3) instructions are used most frequently. However, automatic incrementing by +1 (or decrementing by -1) of HD44780 internal RAM addresses after each data write lessens the MPU program load. The display shift is especially able to perform concurrently with display data write, enabling the user to develop systems in minimum time with maximum programing efficiency. For an explanation of the shift function in its relation to display, see Item 6.6.

When an instruction is executing during internal operation, no instruction other than the busy flag/address read instruction will be executed. Because the busy flag is set to "1" while an instruction is being executed, check to make sure it is on "1" before sending an instruction from the MPU.

(Note) Make sure the HD44780 is not in the busy state (BF=0) before sending the instruction from the MPU to the HD44780. If the instruction is sent without checking the busy flag, the time between first and next instructions is much longer than the instruction time. See Table 4-1 for a list of each instruction execution time.

Table 4.1 Instructions

Instruction	Code										Description	Execution Time(max) (when fcp or fosc is 250 KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DD RAM address 0 in address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire dispaly (D), cursor ON/OFF (C), and blink of cursor position character (B).	40µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and shifts display without changing DD RAM contents.	40µs
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L), and character font (F).	40µs
Set CG RAM Address	0	0	0	1	A_{CG}						Sets CG RAM address. CG RAM data is sent and received after this setting.	40µs
Set DD RAM Address	0	0	1	A_{DD}							Sets DD RAM address. DD RAM data is sent and received after this setting.	40µs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0µs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40µs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40µs
	I/D=1:Increment I/D=0:Decrement S =1:Accompanies display shift. S/C=1:Display shift S/c=0:Cursor move R/L=1:Shift to the right. R/L=0:Shifts to the left. DL=1: 8 bits, DL=0: 4 bits N=1: 2 lines, N=0: 1 line F=1:5×10 dots, F=0:5×7 dots BF=1:Internally operating BF=0:Can accept instruction										DD RAM:Display data RAM CG RAM:Character generator RAM A_{CG}:CG RAM address A_{DD}:DD RAM address. Corresponds to cursor address. AC: Address counter used for both DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fcp or fosc is 270 KHz: $40\mu s \times \frac{250}{270} = 37\mu s$

* No Effect

4.2 Description of Details

(1) Clear Display

Code

RS	R/W	DB7							DB0
0	0	0	0	0	0	0	0	0	1

Writes space code "20" (hexadecimal)(character pattern for character code "20" must be blank pattern) into all DD RAM addresses. Sets DD RAM address 0 in address counter. Returns display to its original status if it was shifted. In other words, the display disappears and the cursor or blink go to the left edge of the display (the first line if 2 lines are displayed). Set I/D=1 (Increment Mode) of Entry Mode. S of Entry Mode doesn't change.

(2) Return Home

Code

RS	R/W	DB7							DB0
0	0	0	0	0	0	0	0	1	*

*Don't care

Sets the DD RAM address 0 in address counter. Returns display to its original status if it was shifted. DD RAM contents do not change. The cursor or blink go to the left edge of the display (the first line if 2 lines are displayed).

(3) Entry Mode Set

Code

RS	R/W	DB7							DB0
0	0	0	0	0	0	0	1	I/D	S

I/D: Increments (I/D=1) or decrements (I/D=0) the DD RAM address by 1 when a character code is written into or read from the DD RAM. The cursor or blink moves to the right when incremented by 1 and to the left when decremented by 1. The same applies to writing and reading of CG RAM.

S : Shifts the entire display either to the right or to the left when S is 1; to the left when I/D=1 and to the right when I/D=0. Thus it looks as if the cursor stands still and the display moves. The display does not shift when reading from the DD RAM nor when writing into or reading out from the CG RAM does it shift when S=0.

(4) Display ON/OFF Control

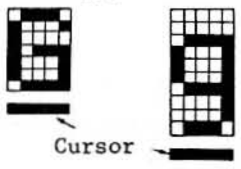
Code

RS	R/W	DB7							DB0
0	0	0	0	0	0	1	D	C	B

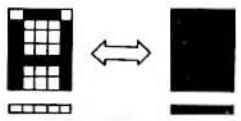
D: The display is ON when D=1 and OFF when D=0. When off due to D=0, display data remains in the DD RAM. It can be displayed immediately by setting D=1.

C: The cursor displays when C=1 and does not display when C=0. Even if the cursor disappears, the function of I/D, etc. does not change during display data write. The cursor is displayed using 5 dots in the 8th line when the 5 x 7 dot character font is selected and 5 dots in the 11th line when the 5 x 10 dot character font is selected.

B: The character indicated by the cursor blinks when B=1. The blink is displayed by switching between all blank dots and display characters at 409.6ms interval when fcp or fosc=250kHz. The cursor and the blink can be set to display simultaneously.
(The blink frequency changes according to the reciprocal of fcp or fosc. $409.6 \times \frac{250}{270} = 379.2$ms when fcp=270kHz.)



5 X 7 dot character font　　5 X 10 dot character font



Alternating display

(a) Cursor Display Example　　(b) Blink Display Example

(5) Cursor or Display Shift

	RS	R/W	DB7							DB0
Code	0	0	0	0	0	1	S/C	R/L	*	*

*Don't care

Shifts cursor position or display to the right or left without writing or reading display data. This function is used to correct or search for the display. In a 2-line display, the cursor moves to the 2nd line when it passes the 40th digit of the 1st line. Notice that the 1st and 2nd line displays will shift at the same time. When the displayed data is shifted repeatedly each line only moves horizontally. The 2nd line display does not shift into the 1st line position.

S/C	R/L	
0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right.(AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Address counter (AC) contents do not change if the only action performed is shift display.

(6) Function Set

	RS	R/W	DB7							DB0
Code	0	0	0	0	1	DL	N	F	*	*

* (Don't Care)

DL : Sets interface data length. Data is sent or received in 8 bit lengths ($DB_7 \sim DB_0$) when DL=1 and in 4 bit lengths ($DB_7 \sim DB_4$) when DL=0.
When the 4 bit length is selected, data must be sent or received twice.

N : Sets number of display lines.

F : Sets character font.

(Note) Perform the function at the head of the program before executing all instructions (except "Busy flag/address read"). From this point, the function set instruction cannot be executed unless the interface data length is changed.

N	F	No. of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5×7 dots	1/8	
0	1	1	5×10 dots	1/11	
1	*	2	5×7 dots	1/16	Cannot display 2 lines with 5×10 dot character font.

* (Don't Care)

(7) Set CG RAM Address

	RS	R/W	DB7							DB0
Code	0	0	0	1	A	A	A	A	A	A

←Higher Order Bits   Lower Order→ Bits

Sets the CG RAM address into the address counter in binary AAAAAA.

Data is then written or read from the MPU for the CG RAM.

(8) Set DD RAM Address

	RS	R/W	DB7							DB0
Code	0	0	1	A	A	A	A	A	A	A

Higher Order Bits   Lower Order→ Bits

Sets the DD RAM address into the address counter in binary AAAAAAA.

Data is then written or read from the MPU for the DD RAM.

However, when N=0 (1-line display), AAAAAAA is "00"∿"4F" (hexadecimal).
When N=1 (2-line display), AAAAAAA is "00"∿"27" (hexadecimal) for the first line, and "40"∿"67" (hexadecimal) for the second line.

(9) Read Busy Flag and Address

	RS	R/W	DB7							DB0
Code	0	1	BF	A	A	A	A	A	A	A

←Higher Order Bits · Lower Order Bits→

Reads the busy flag (BF) that indicates the system is now internally operating by a previously received instruction. BF=1 indicates that internal operation is in progress. The next instruction will not be accepted until BF is set to "0". Check the BF status before the next write operation.

At the same time, the value of the address counter expressed in binary AAAAAA is read out. The address counter is used by both CG and DD RAM addresses, and its value is determined by the previous instruction. Address contents are the same as in Items (7) and (8).

(10) Write Data to CG or DD RAM

	RS	R/W	DB7							DB0
Code	1	0	D	D	D	D	D	D	D	D

←Higher Order Bits Lower Order Bits→

Writes binary 8 bit data DDDDDDDD to the CG or the DD RAM.
Whether the CG or DD RAM is to be written into is determined by the previous specification of CG RAM or DD RAM address setting. After write, the address is automatically incremented or decremented by 1 according to entry mode. The entry mode also determines display shift.

(11) Read Data from CG or DD RAM

	RS	R/W	DB7							DB0
Code	1	1	D	D	D	D	D	D	D	D

←Higher Order Bits Lower Order Bits→

Reads binary 8 bit data DDDDDDDD from the CG or DD RAM.
The previous designation determines whether the CG or DD RAM is to be read. Before entering the read instruction, you must execute either the CG RAM or DD RAM address set instruction. If you don't, the first read data will be invalidated. When serially executing the "read" instruction, the next address data is normally read from the second read. The "address set" instruction need not be executed just before the "read" instruction when shifting the cursor by cursor shift instruction (when reading out DD RAM). The cursor shift instruction operation is the same as that of the DD RAM's address set instruction.

After a read, the entry mode automatically increases or decreases the address by 1. However, display shift is not executed no matter what the entry mode is.

(Note) The address counter (AC) is automatically incremented or decremented by 1 after "write" instructions to either CG RAM or DD RAM. RAM data selected by the AC cannot then be read out even if "read" instructions are executed. The conditions for correct data read out are: execute either the address set instruction or cursor shift instruction (only with DD RAM), just before reading out execute the "read" instruction from the second time the "read" instruction is serial.

5. Electrical Characteristics

5.1 Absolute Maximum Ratings

Item	Symbol	Limit	Unit	Note
Power Supply Voltage (1)	V_{CC}	-0.3 to +7.0	V	
Power Supply Voltage (2)	V1 to V5	V_{CC}-13.5 to V_{CC}+0.3	V	3
Input Voltage	V_T	-0.3 to V_{CC}+0.3	V	
Operating Temperature	Topr	-20 to +75	°C	
Storage Temperature	Tstg	-55 to +125	°C	

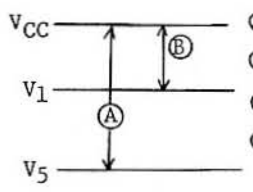
Note 1: If LSI's are used above absolute maximum ratings, they may be permanently destroyed. Using them within electrical characteristic limits is strongly recommended for normal operation. Use beyond these conditions will cause malfunction and poor reliability.

Note 2: All voltage values are referenced to GND=0V.

Note 3: Applies to V1 to V5. Must maintain $V_{CC} \geqq V1 \geqq V2 \geqq V3 \geqq V4 \geqq V5$.
(high ⟵ ⟶ low)

5.2 Electrical Characteristics

V_{CC}=5V±10%, Ta=-20 to +75°C



Ⓐ=$V_{CC}-V_5$
Ⓑ=$V_{CC}-V_1$
Ⓐ≥1.5V
Ⓑ≤0.25xⒶ

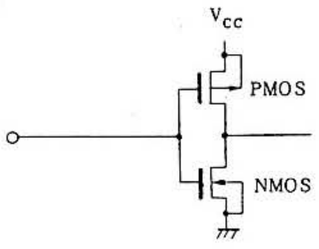
(The conditions of V_1, V_5 voltages are for proper operation of the LSI and not for the LCD output level. The LCD drive voltage condition for the LCD output level is specified in "LCD voltage V_{LCD}".)

Item	Symbol	Test condition	Limit			Unit	Note
			min	typ	max		
Input "High" Voltage (1)	V_{IH1}		2.2	-	V_{CC}	V	(2)
Input "Low" Voltage (1)	V_{IL1}		-0.3	-	0.6	V	(2)
Output "High" Voltage (1)(TTL)	V_{OH1}	$-I_{OH}$=0.205mA	2.4	-	-	V	(3)
Output "Low" Voltage (1) (TTL)	V_{OL1}	I_{OL}=1.2mA	-	-	0.4	V	(3)
Output "High" Voltage (2)(CMOS)	V_{OH2}	$-I_{OH}$=0.04mA	0.9V_{CC}	-	-	V	(4)
Output "Low" Voltage (2) (CMOS)	V_{OL2}	I_{OL}=0.04mA	-	-	0.1V_{CC}	V	(4)
Driver Voltage Descending (COM)	V_{COM}	Id=0.05mA	-	-	2.9	V	(10)
Driver Voltage Descending (SEG)	V_{SEG}	Id=0.05mA	-	-	3.8	V	(10)
Input Leakage Current	I_{IL}	V_{in}=0 to V_{CC}	-	-	1	μA	(5)
Pull up MOS Current	$-I_p$	V_{CC}=5V	50	125	250	μA	
Power Supply Current (1)	I_{CC1}	Ceramic filter oscillation V_{CC}=5V, f_{osc}= 250kHz	-	0.55	0.8	mA	(6)
Power Supply Current (2)	I_{CC2}	Rf oscillation External clock operation V_{CC}=5V, f_{osc}= f_{cp}=270kHz	-	0.35	0.6	mA	(6) (11)
External Clock Operation							
External Clock Frequency	f_{cp}		125	250	350	kHz	(7)
External Clock Duty	Duty		45	50	55	%	(7)
External Clock Rise Time	t_{rcp}		-	-	0.2	μs	(7)
External Clock Fall Time	t_{fcp}		-	-	0.2	μs	(7)
Input "High" Voltage (2)	V_{IH2}		V_{CC}-1.0	-	V_{CC}	V	(12)
Input "Low" Voltage (2)	V_{IL2}		-0.3	-	1.0	V	(12)
Internal Clock Operation (Rf oscillation)							
Clock Oscillation Frequency	f_{osc}	Rf=91kΩ±2%	190	270	350	kHz	(8)
Internal Clock Operation (Ceramic filter oscillation)							
Clock Oscillation Frequency	f_{osc}	Ceramic filter	245	250	255	kHz	(9)
LCD Voltage	V_{LCD1}	$V_{CC}-V_5$ 1/5bias	4.6	-	11	V	(13)
	V_{LCD2}	$V_{CC}-V_5$ 1/4bias	3.0	-	11	V	(13)

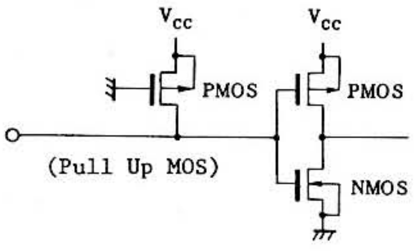
Note 1: The following are I/O terminal configurations ecxept for liquid crystal display output.

- Input Terminal

Applicable Terminals: E
(No pull up MOS)

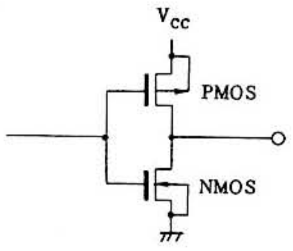


Applicable Terminals: RS, R/W
(With pull up MOS)



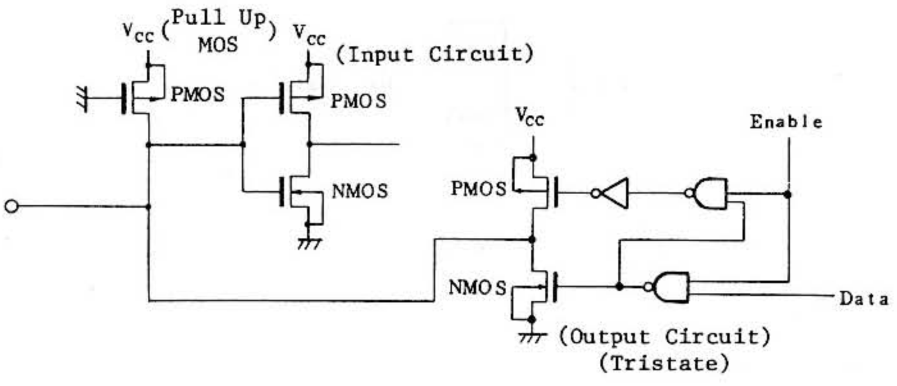
- Output Terminal

Applicable Terminals: CL_1, CL2, M, D



- I/O Terminal

Applicable Terminals: DB_0 to DB_7



Note 2: Input terminals and I/O terminals
Excludes OSC_1 terminals.

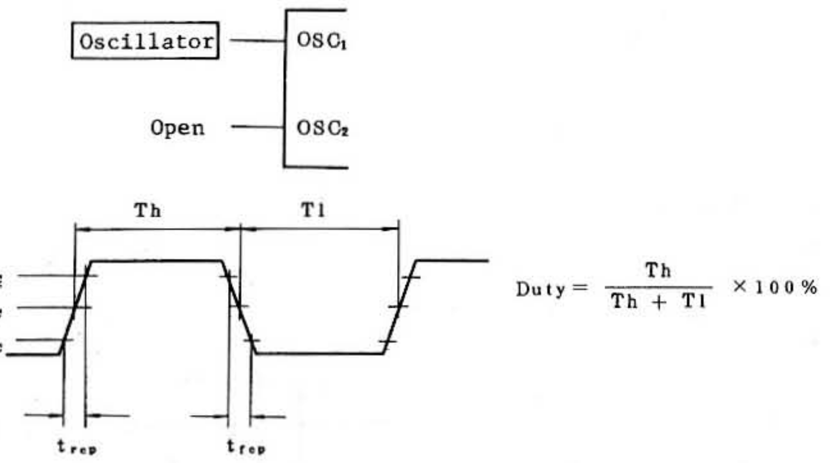
Note 3: I/O terminals.

Note 4: Output terminals.

Note 5: Current flowing through pull-up MOS's and output drive MOS's is excluded.

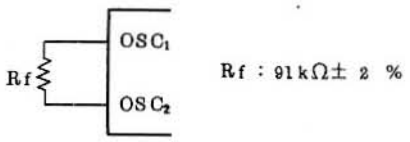
Note 6: Input/Output current is excluded. When input is at the intermediate level with CMOS, excessive current flows through the input circuit to the power supply. To avoid this, input level must be fixed at high or low.

Note 7: External clock operation.



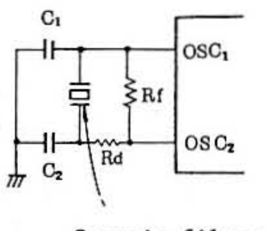
$$\text{Duty} = \frac{Th}{Th + Tl} \times 100\,\%$$

Note 8: Internal oscillator operation using oscillation resistor Rf.



Since oscillation frequency varies depending on OSC_1 and OSC_2 terminal capacity, wiring length for these terminals should be minimized.

Note 9: Internal oscillator operation using a ceramic filter. is used.



Ceramic filter

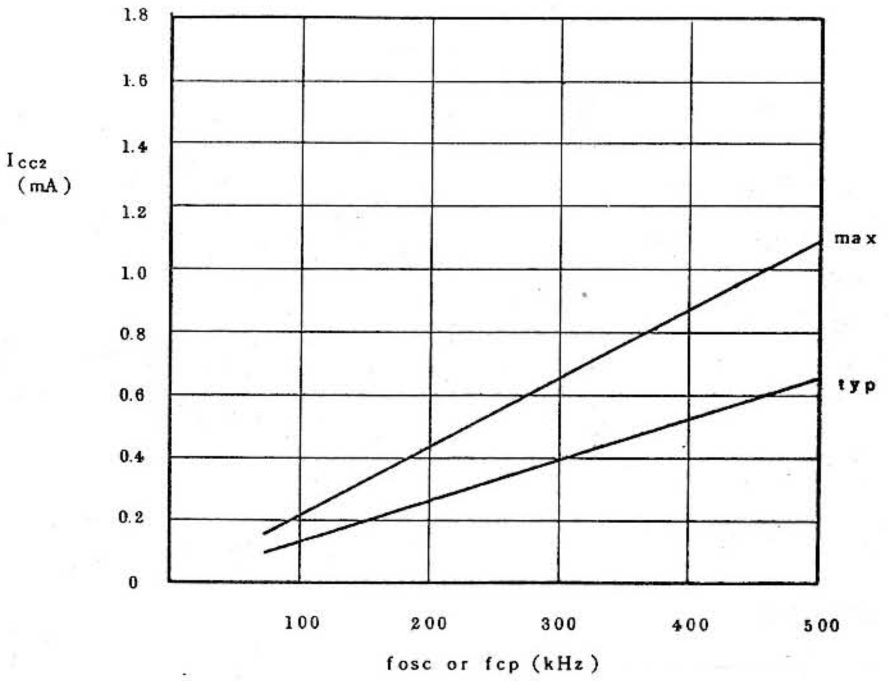
Ceramic filter:	CBS250A (Murata)
Rf:	1MΩ ±10%
C_1:	680pF±10%
C_2:	680pF±10%
Rd:	3.3kΩ±5%

Note 10: Applies to both V_{COM} and V_{SEG} voltage drops.

V_{COM}: From power supply terminal V_{CC}, V1, V4, V5 to each common signal terminal (COM_1 to COM_{16})

V_{SEG}: From power supply terminal V_{CC}, V2, V3, V5 to each segment signal terminal (SEG_1 to SEG_{40})

Note 11: Relation between operation frequency and current consumption is shown in this diagrom. (V_{CC} = 5V)



Note 12: Applied to OSC_1 terminal.

Note 13: The condition for COM pin voltage drop (V_{COM}) and SEG pin voltage drop (V_{SEG}).

5.3 Timing Characteristics

Write Operation

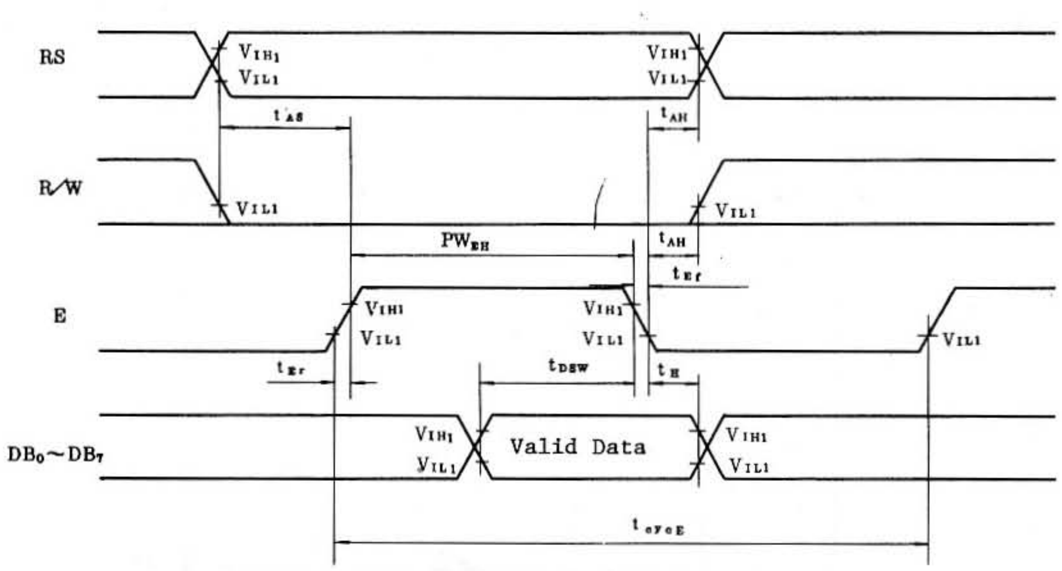


Fig.5-1 Bus Write Operation Sequence
(Writing data from MPU to HD44780)

Read Operation

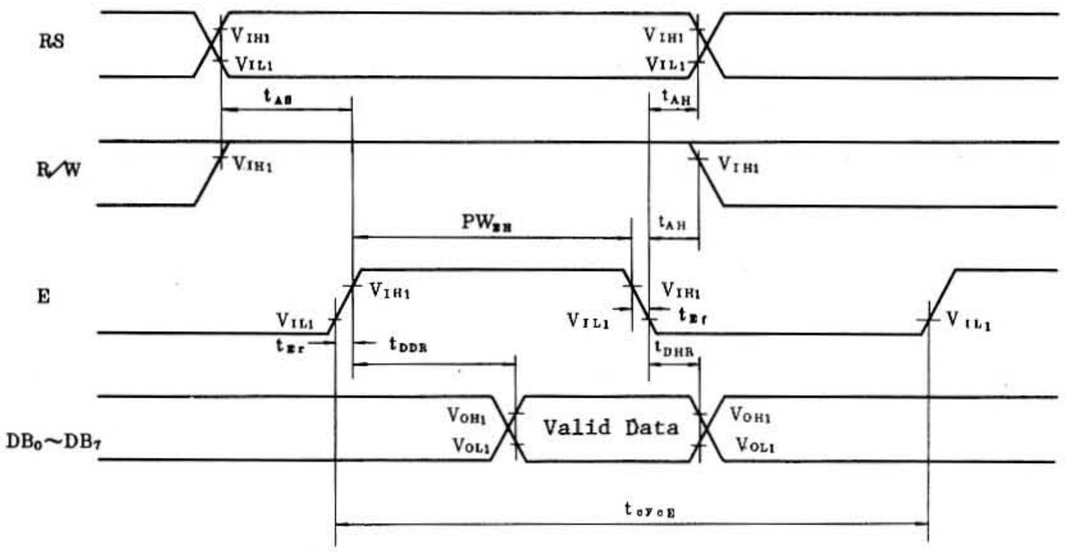


Fig. 5-2 Bus Read Operation Sequence
(Reading out data from HD44780 to MPU)

Interface Signal with Driver LSI HD44100H

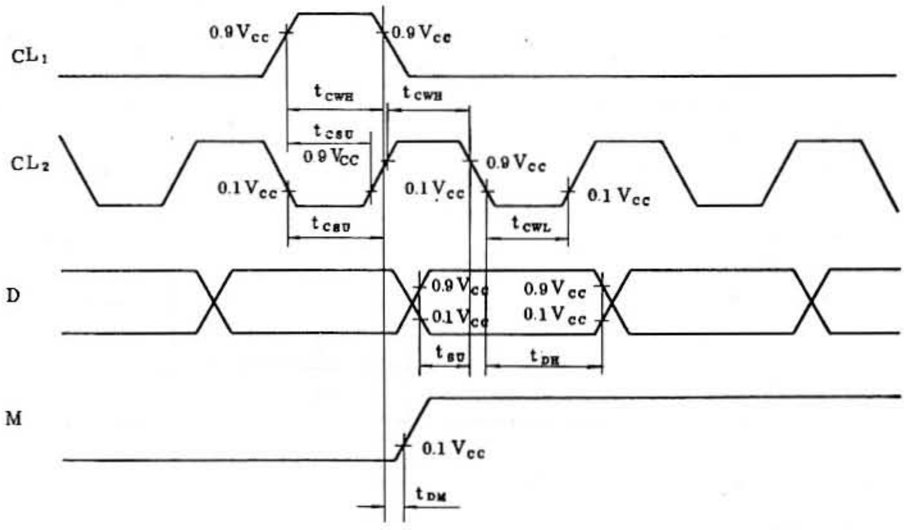


Fig. 5.3 Sending Data to Driver LSI HD44100H

5.3.1 Bus Timing Characteristics ($V_{CC} = 5.0V \pm 10\%$, GND = 0V, Ta = −20 to + 75°C)

Write Operation (Writing data from MPU to HD44780)

Item		Symbol	Test Conditions	Limit		Unit
				min	max	
Enable Cycle Time		t_{cycE}	Fig. 5.1	1000	-	ns
Enable Pulse Width	"High" level	PW_{EH}	Fig. 5.1	450	-	ns
Enable Rise/Fall Time		t_{Er}, t_{Ef}	Fig. 5.1	-	25	ns
Address Set-up Time	RS, R/W —E	t_{AS}	Fig. 5.1	140	-	ns
Address Hold Time		t_{AH}	Fig. 5.1	10	-	ns
Data Set-up Time		t_{DSW}	Fig. 5.1	195	-	ns
Data Hold Time		t_H	Fig. 5.1	10	-	ns

Read Operation (Reading data from HD44780 to MPU)

Item		Symbol	Test Conditions	Limit		Unit
				min	max	
Enable Cycle Time		t_{cycE}	Fig. 5.2	1000	-	ns
Enable Pulse Width	"High" level	PW_{EH}	Fig. 5.2	450	-	ns
Enable Rise/Fall Time		t_{Er}, t_{Ef}	Fig. 5.2	-	25	ns
Address Set-up Time	RS, R/W —E	t_{AS}	Fig. 5.2	140	-	ns
Address Hold Time		t_{AH}	Fig. 5.2	10	-	ns
Data Delay Time		t_{DDR}	Fig. 5.2	-	320	ns
Data Hold Time		t_{DHR}	Fig. 5.2	20	-	ns

5.3.2 Interface Signal with HD44100H Timing Characteristics

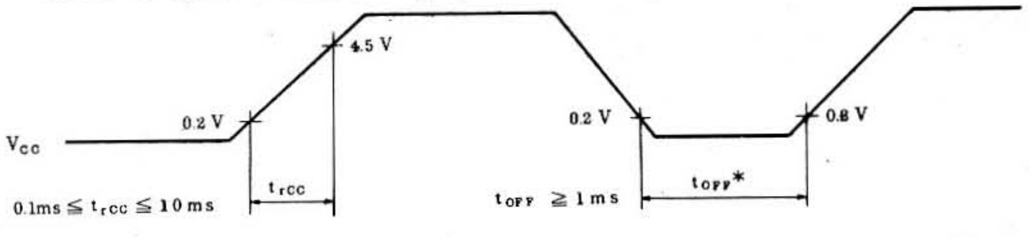
$$\left(\begin{array}{l} VCC = 5.0V \pm 10\%,\ GND = 0V \\ Ta = -20 \text{ to } +75°C \end{array}\right)$$

Item		Symbol	Test Conditions	Limit		Unit
				min	max	
Clock Pulse Width	"High" level	t_{CWH}	Fig. 5.3	800	-	ns
Clock Pulse Width	"High" level	t_{CWL}	Fig. 5.3	800	-	ns
Clock Set-up Time		t_{CSU}	Fig. 5.3	500	-	ns
Data Set-up Time		t_{SU}	Fig. 5.3	300	-	ns
Data Hold Time		t_{DH}	Fig. 5.3	300	-	ns
M Delay Time		t_{DM}	Fig. 5.3	-1000	1000	ns

5.4 Power Supply Conditions Using Internal Reset Circuit

Item	Symbol	Test Conditions	Limit		Unit
			min	max	
Power Supply Rise Time	trcc	-	0.1	10	ns
Power Supply OFF Time	t_{OFF}	-	1	-	ns

Since the internal reset circuit will not operate normally unless the preceding conditions are met, initialize by instruction.
(Refer to 3.7.2 "Initializing by Instruction")



(Note) t_{OFF} stipulates the time of power OFF for power supply instantaneous dip or when power supply repeats ON and OFF.

Appendix D

References

8Ø86/8Ø88 16-Bit Microprocessor Primer;
Christopher L. Morgan & Mitchell Waite;
Intel Corporation

8Ø86/8Ø87/8Ø88 Macro Assembly Language Reference Manual;
Intel Corporation

8Ø88 Assembler Language Programming: The IBM PC;
David C. Willen & Jeffrey I. Krantz; Computer Applications
Unlimited; Howard W. Sams & Company

THE 8Ø86 BOOK includes the 8Ø88;
Russell Rector - George Alexy;
OSBORNE/McGraw-Hill

MS-DOS Operating System Macro Assembler Manual;
Microsoft Corporation

Assembly Language Programming for the IBM Personal Computer;
David J. Bradley;
Prentice-Hall, Inc.

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.: M8802-8605C