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 direc-

tive 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.

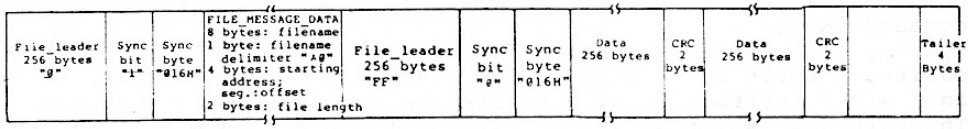
An 8-byte memory space is reserved for the characters which make up the filename. If less than eight characters are used, FILE_NAME_FILLER will fill the unused memory space with the ASCII code for the space character (2ØH) and execute a RET to the main program to execute W_CMD_2. W_CMD_2 will place the 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 0.2 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 0.2 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 0.2 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 tape 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

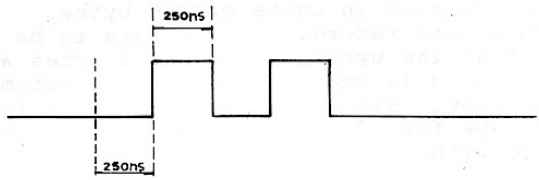
After the whole memory range is output to tape, a file tailer will be output to tape by WRITE_TAILER. The file tailer consists of four bytes of 1.

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_250

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_250 (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 250 ns half cycle and a low 250 ns half cycle as illustrated below:



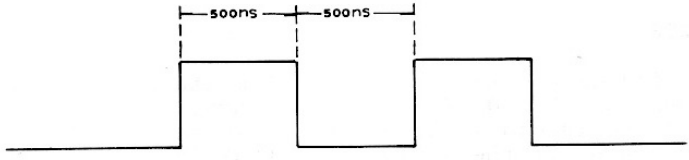
PULSE WAVE FOR A BIT Ø

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 8088 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.