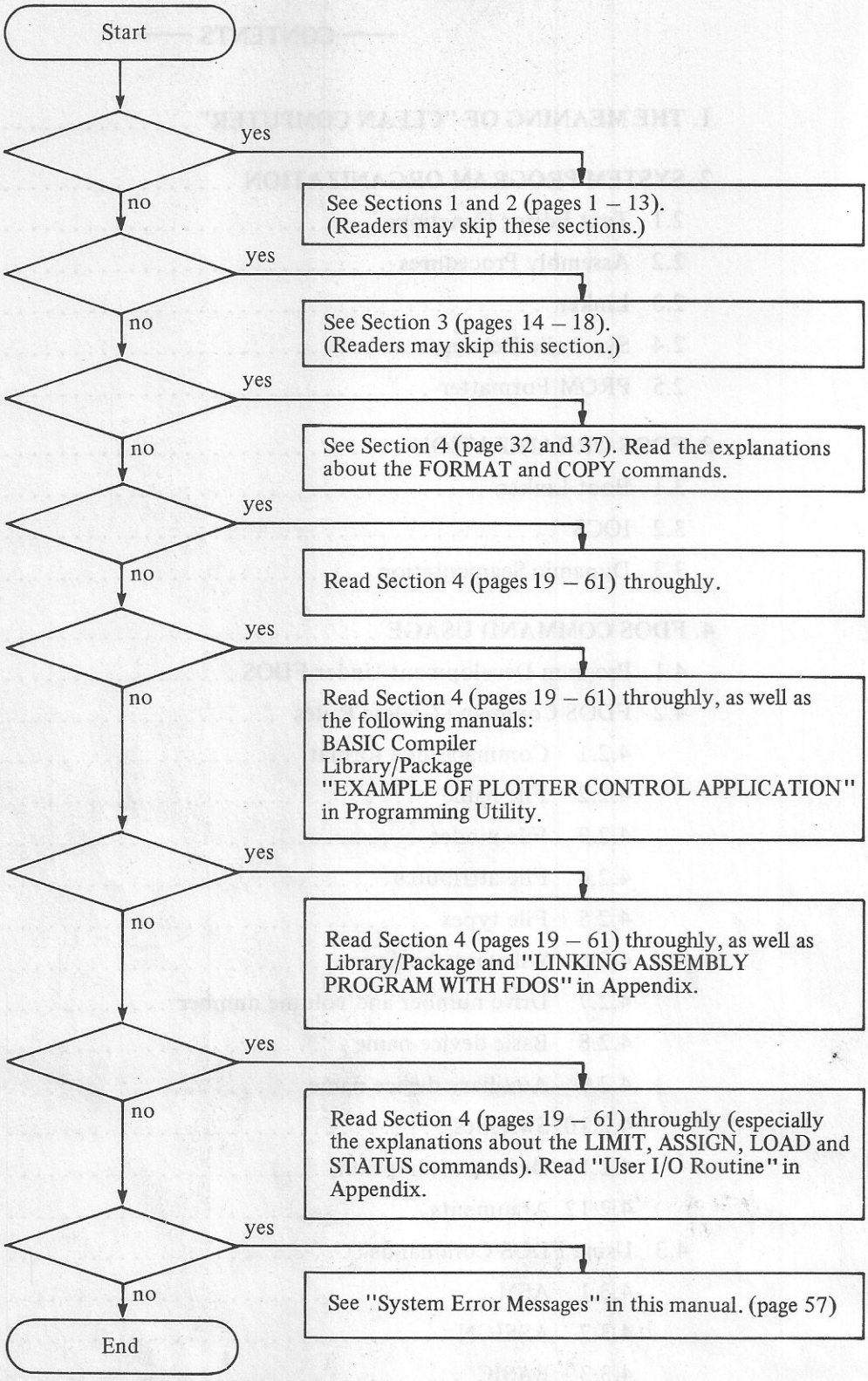
GUIDE TO USE OF THIS MANUAL



—— CONTENTS ——

1. THE MEANING OF "CLEAN COMPUTER"

Three important developments accompanied the shift from the boom in microcomputer kits to the entrance of personal computers.

(1) Mass production reduced the cost of RAM and ROM devices so that they became readily available.

This development eliminated the need to devote great amounts of time and effort to compressing system functions to the maximum extent possible to conserve valuable memory for user programs. Now it is more important that system programs be written and managed in a structured manner and that their overall usefulness be raised. It is more and more apparent that what the user comes in contact with is not so much a unit of hardware as a software reinforced computer.

(2) Compact, reliable external memory units with large storage capacities became available.

Floppy disks and fixed disks are currently the basis for system configurations, but sooner or later charge coupled devices and magnetic bubble memories will be used in this capacity. This suggests that there will be increasing stratification of programs culminating in operating systems, and that the efficiency of systems will also increase. From the user's point of view, this means that a wide variety of programs will be readily available for use.

(3) The development of various peripheral circuit LSIs has made possible realization of efficient interfaces with high performance terminals.

This means the main concern of the user in the future will be with how many functions are provided in a system and how useful they are. In terms of the contents of the system, the main concern will be in developing operating systems capable of organically combining terminals and program processing with a minimum of effort on the part of the user. It is even possible that real time processing of multiple tasks and jobs on a level approaching that of minicomputers will become possible with the operating systems of microcomputers.

As is apparent, it is extremely difficult to predict the extent to which computers will evolve as integrated circuit technology and program language theory become widely dispersed. This tends to undermine the belief which some people have that rapid changes in hardware result in good computers.

Although the name "clean computer" has been given to the MZ-80 series, computers are basically clean in principle. As the field of personal computers opens, the concept of embedding a single language, BASIC, in ROM has become a hindrance to use of full computer capacity. Out of consideration for the many different types of service which will be required by users as yet-to-be developed technology comes into use in the future, it will be necessary to preserve the cleanliness of the computer to the maximum degree possible to minimize constraints placed on its use. The ultimate ends to which computers are applied will be determined by the junction of technological possibilities and user requirements; the only other limits imposed are those which are inherent in the fact that the computer is nothing more than a machine. In order for computers and users to get along well together, it is necessary that computers be designed with a minimum of constraints so that they can be suited to user requirements, rather than the other way around. In other words, the usefulness of the computer and the efficiency of the service it provides depends on how clean it is.

The explanations in these publications are intended to show how flexible the MZ-80 series of computers is in terms of system development. A tape-based program development system is provided to enable inexpensive development of small programs; the floppy disk operating system (FDOS) was developed to assist with the creation of large programs which require large quantities of memory. The functions and configuration of FDOS are suited to a range of applications approaching those provided by a low level minicomputer. We think that the software technology and utilization procedures applied in this system will open a new world of possibilities for personal computers.

2. SYSTEM PROGRAM ORGANIZATION

SHARP MZ-80 series system programs include an assembler, a text editor, a linker and a symbolic debugger. They are organized to execute a sequence of assembly phases.

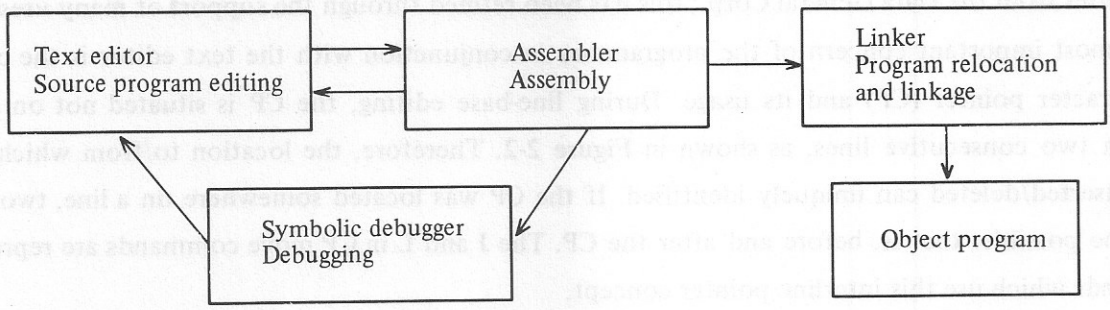


Fig. 2-1 Assembly phases

Figure 2-1 shows the assembly process, which consists of creating source programs, assembling them, relocating and linking the assembly output and debugging them.

One cycle of the phases in the left half of the figure makes up a program creation stage. The programmer prepares a source program with the text editor and creates a source file, then inputs it to the assembler. The assembler analyzes and interprets the syntax of the source program and assembly language instructions into relocatable binary code. When the assembler detects errors, it issues error messages. The programmer then corrects the errors in the source program with the text editor.

After all assembly errors are corrected, the programmer inputs the relocatable program (the relocatable binary file), output by the assembler to the symbolic debugger. The symbolic debugger reads the object program into the link area in an executable form and runs the program. During the debugging phase, the programmer can set breakpoints in the program to start, interrupt and continue program execution, and to display and alter register and memory contents for debugging purposes. If program logic errors and execution inefficiency are detected during the debugging phases, the programmer reedits the source program using the text editor.

After all bugs are removed from the source program, the programmer loads and links the program unit(s) in the relocatable file(s) and creates an object program in executable form with the linker.

Each system program always generates an output file for use in other system programs. **Figure 4-1** shows the interrelationship of the system programs.

As shown above, the program development phases are executed by four independent system programs. By assigning the system functions to separate programs, the MZ-80 series can accomodate large-scale, serious application programs, thus enhancing its program development capabilities. "PROM formatter" is provided which punches object programs into paper tape in several formats for use with various PROM writers now on the market.

The system program commands are listed in the last part of Appendix.

2.1 Text Editor Functions

The major functions of a text editor are to insert, delete and modify characters, words and/or lines. If the editor does not allow the programmer to use these functions interactively and easily, he will have to devote more effort to editing and modifying programs than to executing them. To alleviate this problem, SHARP uses a command format which is almost perfectly compatible with that of the NOVA minicomputer series from the Data General Corp.; this has been refined through the support of many uses.

The most important concern of the programmer in conjunction with the text editor is the concept of the character pointer (CP) and its usage. During line-base editing, the CP is situated not on a line but between two consecutive lines, as shown in **Figure 2-2**. Therefore, the location to/from which a line is to be inserted/deleted can uniquely identified. If the CP was located somewhere on a line, two locations would be possible; that is, before and after the CP. The J and L in CP move commands are representative commands which use this interline pointer concept.

During character-base editing, the CP is situated not on a character but between two consecutive characters. This permits close editing. The programmer will become accustomed to the text editor quickly if he is aware of what commands use the interline CP and what command use the intercharacter CP concept.

During normal editing sessions, several commands are combined to carry out an intended task. Such commands can be placed on a line separated by separators so that the programmer lists them as they come into his head.

The delimiter " ➡ " is entered when [CTRL] + [Z] are pressed.

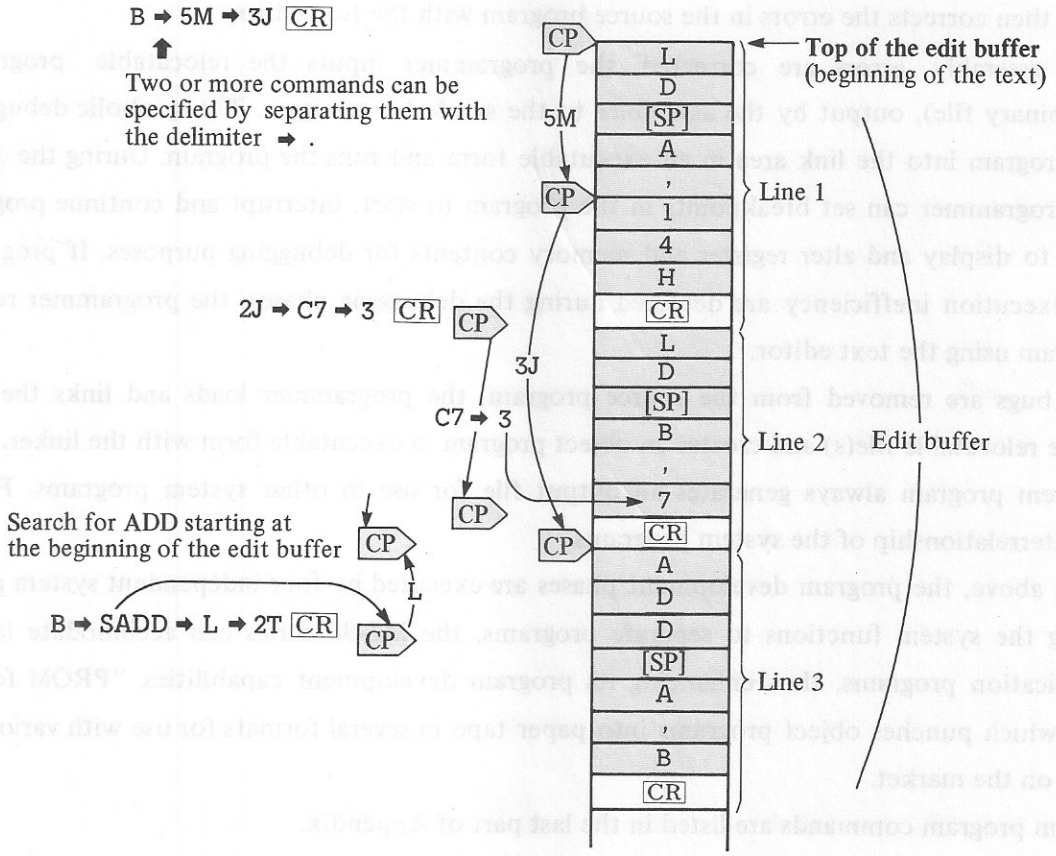


Fig. 2-2 Character pointer movement

2.2 Assembly Procedures

As the programmer becomes familiar with the Z-80 instructions, he is able to construct programs more easily, even though he may feel difficulty in grasping the structure of large programs. At this stage, it is not hard for the programmer to handle other microprocessors such as the M6800 and the F-8 with the help of good reference manuals. One of the major reasons for this is the operating principles and architecture of most computers tend to be alike. It is therefore possible to develop a general purpose assembler for such micro-processors. In this section, the technique employed in the MZ-80 assembler is described. This will serve as a model for designing general-purpose assemblers.

The basic operation of any assembler is the interpretation of statements. It is therefore important to establish a proper statement coding format. **Figure** 2-3 shows an example of a coding format, used in the MZ-80 assembler, which is familiar to humans and which is easy for the computer to interpret.

Scanning the statements in this format, the assembler:

(1) Recognizes labels and stores them into the label table,

(2) Recognizes fields and assembles object codes,

(3) Generates an assembly listing, and

(4) Generates relocatable binary code.

Step (2) differs from one processor to another. The assembler constitutes a general-purpose assembler if it can perform this step flexibly. As the nucleus of the process for step 2, an instruction list **(Figure 2-4)** and a 2-dimensional operation table **(Table 1)** are introduced.

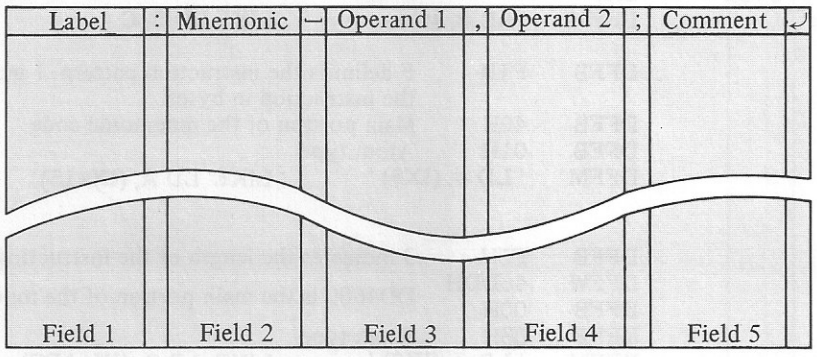


Fig. 2-3 Assembler coding format

The symbol # in the instruction list represents a register and the symbol $ represents a label or numeric value. The assembler identifies each instruction by matching the read assembly statement with this listing. As a result of this match, the assembler produces the major portion of the op-code, the byte length of the instruction and its atom type. An atom type is one of the numbers identifying the instruction groups of the Z-80 instruction set. As is seen from **Table 1**, there are 48 atom types; these are sufficient for newly defined instructions.

The operations to be performed for each atom type are designated by a 16-bit flag field. For atom type 01, for example, flag bits 0, 3 and 4 are set, indicating that the operations identified by these bits are to be performed in that order. The control words identified by the set flag bits specify the actual operations to be performed. Flag 3 indicates that this instruction must be a 1-byte instruction, that it must shift the data to the left 3 bits, and that the size of the field must be 3 bits or less. Similarly, flag 4 indicates that this atom type represents the LD r,r' operation.

Let us examine atom type 18. The set flag bits are 0, 1 and A. The control word for flag 1 is all zeros, which means no operation. Flag A indicates that the instruction requires address modification (address procedure) and that the address field must be not longer than 16 bits (size of the field). Thus, atom type 18 represents instructions such as JP nn' and JP NZ, nn'.

The above assembler operating procedure is summarized in **Figure 2-5**. Most of the assembly operations involve table references. In fact, the assembler uses a register table, a separator table and a label table during the assembly process, in addition to the instruction list and the 2-dimensional operation table. If these tables are redefined to conform to a new instruction set the assembler may also be used as a cross assembler.

```
01  0000                :
02  0000                : INSTRUCTION LIST
03  0000                :
04  0000                SYMP :    ENT
05  0000  4C442023                DFFM    'LD #, #'          ;LIKE LD B, C
06  0004  2C23
07  0006  F1                      DFFB    F1H        F delimits the instruction pattern. 1 indicates the length of
                                                     the instruction in bytes.
08  0007  40                      DFFB    40H        Main portion of the mnemonic code
09  0008  01                      DFFB    01H        Atom type
10  0009  4C442023                DFFM    'LD #, (IX$)'      ;LIKE  LD A, (IX+15)
11  000D  2C284958
12  0011  2429
13  0013  F3                      DFFB    F3H        3 indicates the length of the instruction in bytes.
14  0014  DD46                    DFFW    46DDH      DD4600 is the main portion of the mnemonic code.
15  0016  00                      DFFB    00H
16  0017  03                      DFFB    03H        Atom type
17  0018  4C442023                DFFM    'LD #, (IY$)'      ;LIKE  LD B, (IY+AFC)
18  001C  2C284959
19  0020  2429
20  0022  F3                      DFFB    F3H
21  0023  FD46                    DFFW    46FDH
22  0025  00                      DFFB    00H
23  0026  03                      DFFB    03H
24  0027  4C442028                DFFM    'LD (IX$), #'      ;LIKE  LD (IX+23), A
25  002B  49582429
26  002F  2C23
27  0031  F3                      DFFB    F3H
28  0032  DD70                    DFFW    70DDH
29  0034  00                      DFFB    00H
30  0035  04                      DFFB    04H
```

Fig. 2-4 Instruction list (part)

Table 1 Two-dimensional operation table

Atom	Description	Flags (analyzed and processed in ascending flag bit number order) 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Reserved																
01	LD #, #	1			1	1											
02	LD #, $	1					1			1							
03	LD #, (IX+$) LD #, (IY+$)	1					1	1		1							
04	LD (IX+$), # LD (IY+$), #	1	1							1	1						
05	LD (IX+$), $ LD (IY+$), $	1	1						1	1							
06	LD A, ($)	1	1									1					
07	LD ($), A	1										1					
08	LD BC, $ etc.	1	1									1					
09	LD IX, $ LD IY, $	1	1									1					
0A	LD HL, ($)	1	1									1					
0B	LD BC, ($) etc.	1	1									1					
0C	LD ($), HL	1										1					
0D	LD ($), BC etc.	1										1					
0E	ADD A, # etc.	1	1			1											
0F	ADD A, $ etc.	1	1							1							
10	ADD A, (IX+$) etc.	1	1					1		1							
11	INC # etc.	1			1												
12	INC (IX+$) etc.	1	1							1							
13	RLC # etc.	1				1											
14	RLC (IX+$) etc.	1	1						1								
15	BIT $, # etc.	1		1		1											
16	BIT $, (HL) etc.	1		1													
17	BIT $, (IX+$) etc.	1		1				1	1								
18	JP NZ, $ etc.	1	1									1					
19	JR C, $ etc.	1	1										1				
1A	JR $ DJNZ $	1											1				
1B	SUB # etc.	1				1											
1C	SUB $ etc.	1								1							
1D	SUB (IX+$) etc.	1						1		1							
1E	RST $	1		1													
1F	IN A, ($)	1	1							1							
20	IN #, (C)	1			1												
21	OUT ($), A	1								1							
22	OUT (C), #	1	1		1												
23																	
24																	
2E																	
2F																	
CONTROL WORD	ADDRESS PROCEDURE			1					1	1		1	1				
	MUST BE SINGLE			1	1	1	1		1	1	1		1				
	MUST BE ADR-2												1				
	LEFT SHIFT POSITION			1	1		1										
				1	1		1										
							1		1		1						
	DON'T CARE																
	EQUATION PROCEDURE			1					1	1							
	SIZE OF FIELD			1	1	1	1				1						
				1	1	1	1				1						
									1	1			1				
												1					

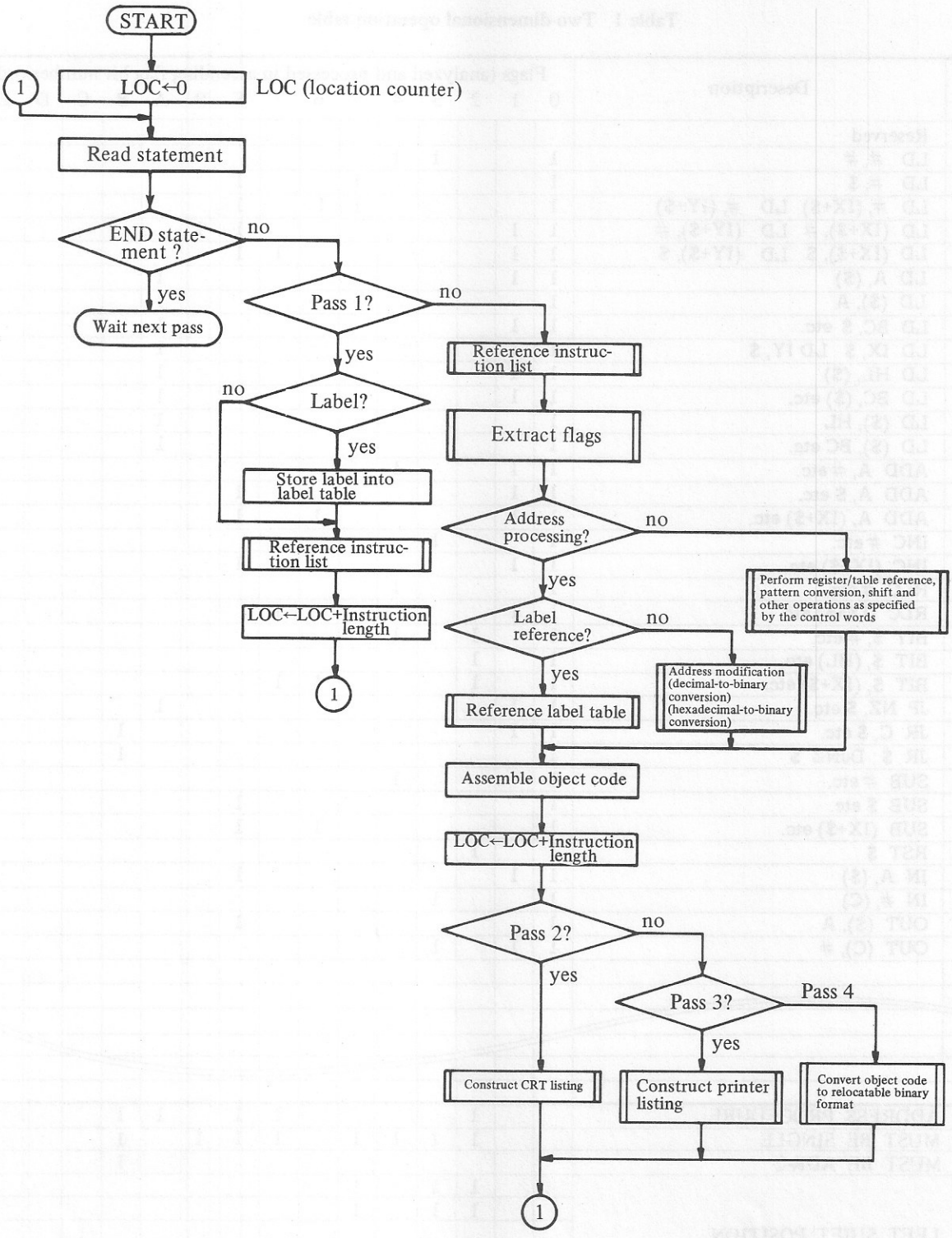


Fig. 2-5 General assembly flow (excluding assembler directive processing)

2.3 Linker

The linker loads and links two or more program units using external symbol referencing instruction from relocatable files and generates absolute binary code in the link area and saves it into an object file. The relocatable files contain control frames and external symbol information. The linker resolves external symbol references and relocates the program units as described below.

(1) External symbol reference resolution

The linker refers to the symbol table when resolving external symbol references (see **Figure 2-6**). The symbol table contains a 9-byte symbol table entry for each external symbol. A symbol table entry consists of a 6-byte field containing the symbol name, a 1-byte field containing the definition status, and a 2-byte field containing an absolute address with which the symbol is defined or a relocation address.

When the linker encounters an external symbol reference while loading the program unit from a relocatable file, it checks to determine whether the symbol has been cataloged in the symbol table.

(1) If it has not been cataloged, the linker enters it into the symbol table as a new undefined symbol, loads the relocation address into the symbol table entry and loads code FFFFH into the operand address of the instruction in memory.

(2) If it has been cataloged and defined, the linker loads the defined absolute address into the operand address in memory.

(3) If it has been cataloged but not defined, the linker moves the old relocation address in the symbol table entry to the operand address in memory and loads the new relocation address into the symbol table entry.

Thus, the linker chains undefined references to each symbol and, when the symbol is defined, replaces all reference addresses with the defined absolute address. In other words, when an external symbol defined by the ENT assembler directive appears in the control frame, the linker enters the symbol into the symbol table as a defined symbol and replaces all preceding operand addresses chained in memory (terminated by FFFFH) with the absolute address defined. The programmer can examine the definition status of the symbols using the table dump command.

An example of external symbol reference resolution follows. Assume that three program units are to be linked and that each unit references subroutine SUB1 in the third program unit (see **Figure 2-8**).

When the first CALL SUB1 instruction is encountered in program unit 1, the linker enters SUB1 into the symbol table as an undefined symbol, loads the operand address (relocation address 5001H in this case) into which the value of the symbol is to be loaded into the 2-byte value field of the symbol table entry and loads the code FFFFH into the operand address in memory (see **Figure 2-8(a)**).

When the CALL SUB1 instruction is encountered twice in program unit 2, the linker chains together their operand addresses which reference SUB1 (see **Figure 2-8(b)**). When SUB1 is defiend in program unit 3, the linker designates SUB1 as a defined symbol and loads all operand addresses referencing SUB1 with the defining absolute address. The end of the operand address chain is identified by the code FFFFH. **Figure 2-8(c)** shows that SUB1 is defined by absolute address 5544H. When the linker subsequently encounters a CALL SUB1 instruction, it immediately loads 5544H into the operand address of the instruction since symbol SUB1 has been defined.

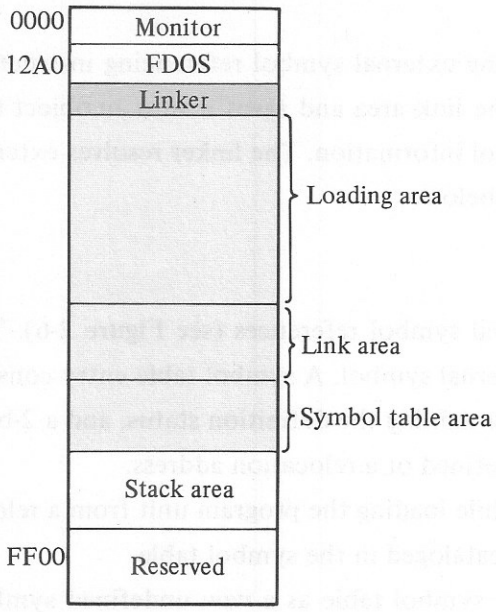


Fig. 2-6 Memory map for the linker

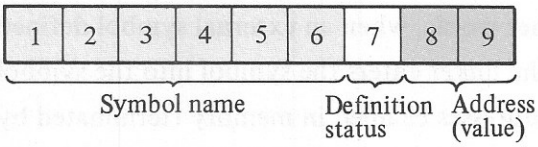


Fig. 2-7 Symbol table entry format

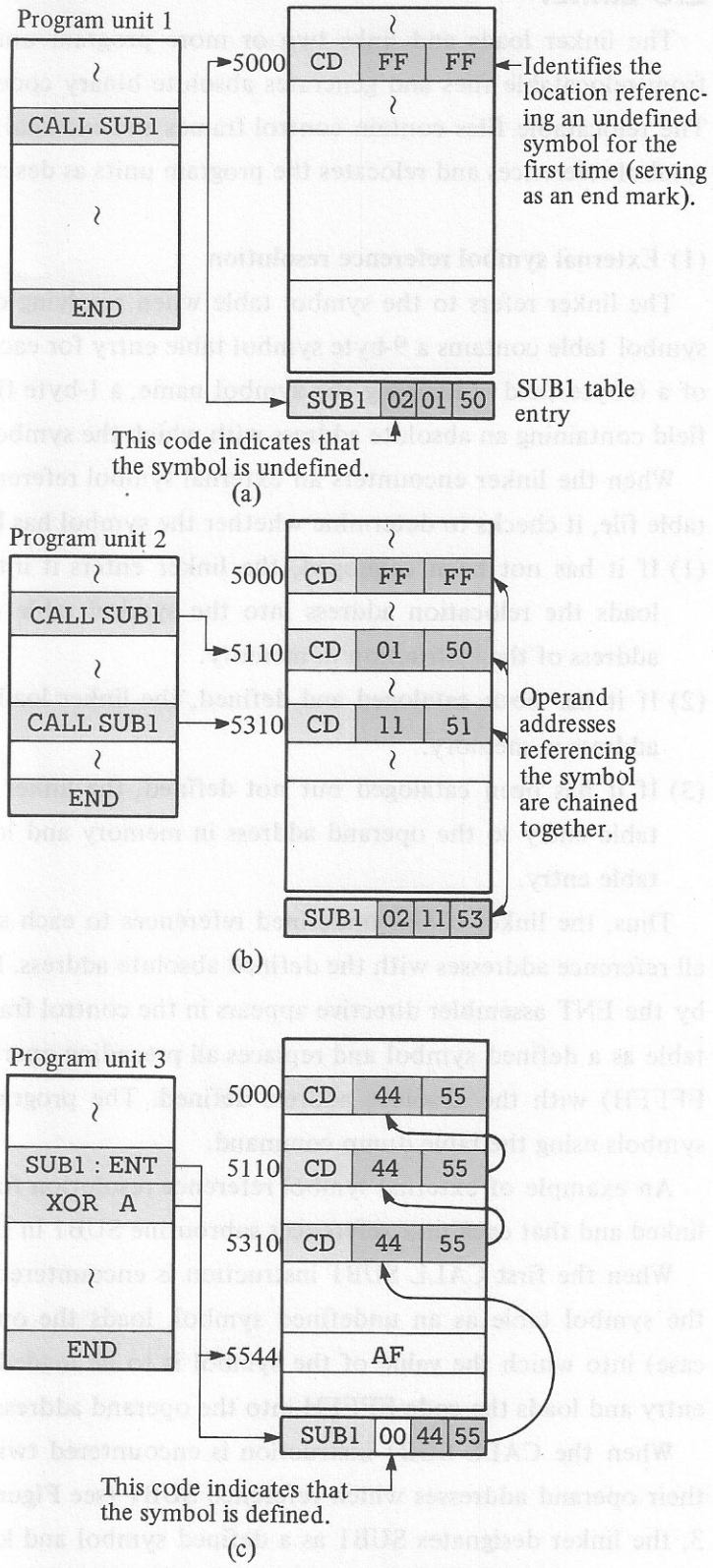


Fig. 2-8 Example of external symbol reference chaining

(2) Program relocation

The linker relocates instructions referencing external symbols while linking the programs. For instructions which reference internal symbols and for which relocation addresses are generated by the assembler, however, the linker produces absolute addresses for the symbols by adding bias to the relocation addresses.

Thus, the linker generates absolute binary code in the link area in an executable format which is dependent on the bias specified by the programmer when the program unit is loaded. When creating an object file, the linker saves the absolute binary code from the link area in the file together with its loading address and execution address.

2.4 Symbolic Debugger

The symbolic debugger inputs relocatable files under the same input conditions as the linker except that it presumes that absolutable binary code is loaded into the link area in an immediately executable form. The symbolic debugger permits the programmer to debug his program while running it.

With the symbolic debugger, the programmer can run a program, interrupts its execution at specified locations and check the system status at these points. The programmer specifies the breakpoints at which program execution is interrupted. When a breakpoint is encountered, the symbolic debugger saves the operation code at the address set as the breakpoint in the break table and replaces it with an RST 7 instruction (FFH) (see **Figure 2-9**).

The RST 7 instruction is a 1-byte call instruction to address 38H in hexadecimal. Its operation is as follows:

$(SP - 1) \leftarrow PC_H$, $(SP - 2) \leftarrow PC_L$

$PC \leftarrow 0038H$

Hexadecimal address 38H (in monitor ROM) contains a JP 1038H instruction which transfers control to the breakpoint control routine in the debugger.

Each breakpoint is associated with a break counter. A break is actually taken when the breakpoint is reached the number of times specified by the break counter. Before the break count is reached, execution is continued with the original operation code saved.

When a break occurs, the debugger saves the contents of the CPU registers in the register buffer and displays them in the screen. When the program is restarted, the debugger restores the contents of the register buffer to the CPU registers and pops the break address.

The programmer can specify a maximum of nine breakpoints and a maximum break count of 14 in decimal.

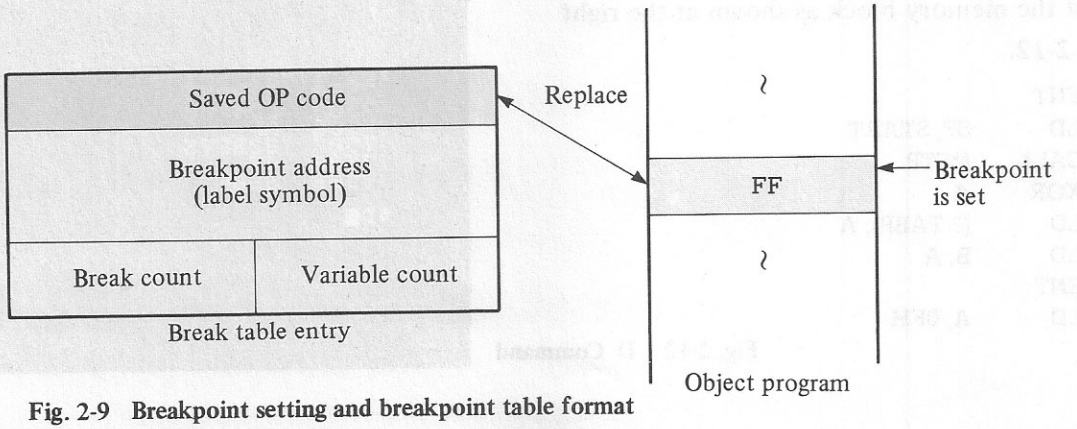
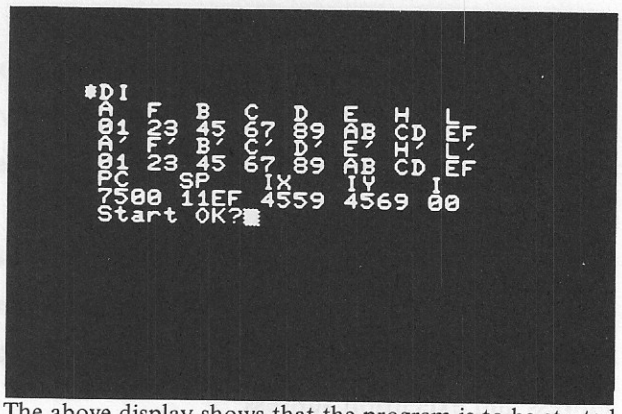


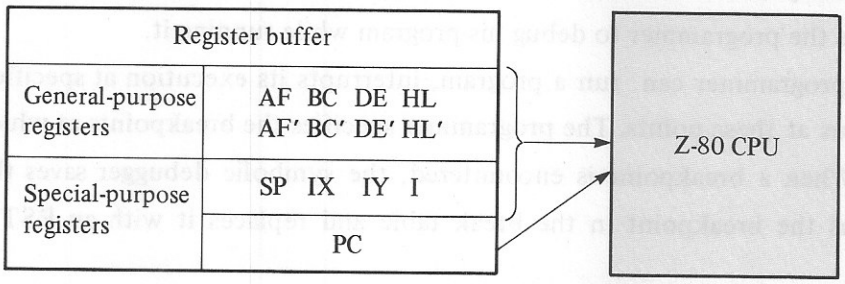
Fig. 2-9 Breakpoint setting and breakpoint table format

The symbolic debugger has indicative start and memory list dump commands in addition to the breakpoint setting command, execution command, memory dump command and register command. The indicative start (I) command displays contents of the CPU registers with which the program is to be executed for confirmation before actually transferring control to the address designated by the program counter (PC) displayed. For example, when an I command is enterd, the display shown in **Figure 2-10** appears on the screen. When the programmer pressed [CR] after confirming the CPU register contents, the debugger initiates an indicative start as shown in **Figure 2-11**.



The above display shows that the program is to be started at address 7500 (hex) with the CPU register values shown.

Fig. 2-10 I command example



The debugger restores the contents of the general-purpose registers and special-purpose registers SP, IX, IY and I, then the value of the PC and initiates program execution.

Fig. 2-11 I command operation

The memory list dump (D) command displays the machine code in the specified memory block with one instruction on each line.

The symbolic debugger permits the programmer to symbolically specify addresses as shown in **Figure 2-12**. With symbolic addresses, the programmer can specify any addresses in the program wherever the program is located in memory.

The programmer can specify the following types of addresses symbolically:

(1) Addresses represented by a symbol

(2) The address of an instruction 1 to 65535_{10} lines away from the address represented by the symbol

(3) An address ±1 to 65535_{10} bytes away from the address represented by the symbol

Of course, the programmer can also specify memory locations with absolute addresses.

For example, the program unit whose source program is shown at the left of **Figure 2-12** is loaded into memory by the debugger starting at hexadecimal address 7500, execution of a D command will display a dump of the memory block as shown at the right in **Figure 2-12**.

```
START : ENT
        LD      SP, START
        CALL    MSTP
        XOR     A
        LD      (? TABP), A
        LD      B, A
MAIN0 : ENT
        LD      A, 0FH
```

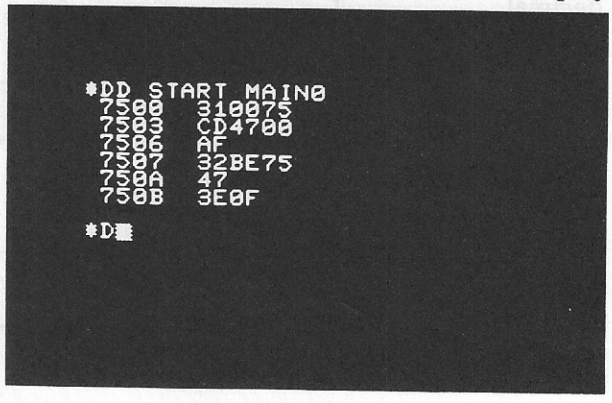


Fig. 2-12 D Command

2.5 PROM Formatter

The PROM formatter generates formatted absolute binary code and stores it into paper tape under the PTP control. It is the system backup software used to transfer object programs to the PROM writer. Currently, the following paper tape output formats are supported (see **Figure 2-13**):

(1) BNPF format: Britronics, Intel and Takeda

(2) B10F format: Takeda

(3) Hexadecimal format: Britronics, Takeda, Minato Electronics

(4) Binary format: Britronics

The variety of tape formats supported by the SHARP PROM formatter extends the application range of programmable ROMs.

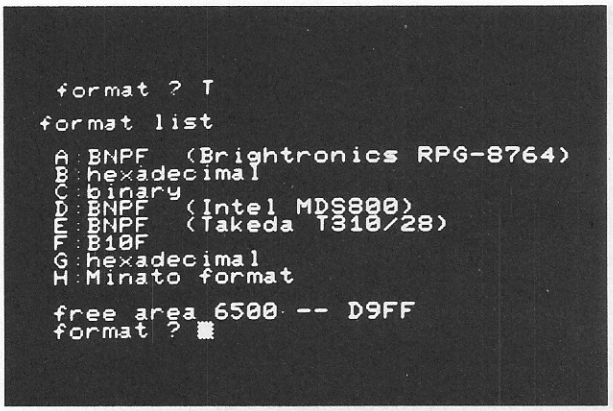


Fig. 2-13 Paper tape output formats

The PROM formatter is made up of format, the PTP and the PRT controls (See **Figure 2-14**). These enable the programmer to perform foramt conversion.

The formatter checks parity in one of three modes (even parity, odd parity or no parity) when reading paper tape. In the formats using ASCII code (BNPF, B10F and hexadecimal), the most significant bit is assigned even or odd parity. When even parity is used, for example, ASCII code "A" (41 hexadecimal) is punched as is, whereas "C" (43 hexadecimal) is converted to C3 in hexadecimal before being punched by setting its MSB. The parity mode can be set using the P command with the desired switch assigned, e.g. ＊P$PTP/PE/LF.

This PROM formatter assumes that the PTP/PTR interface is compatible with the RP-600 puncher/reader from the Nada Electronics Laboratory. It can control RP-600 directly using the general-purpose I/O card (MZ-80IO2). It can also control other models, such as the DPT26A paper tape punch from Anritsu, if I/O conforming to the punch specifications can be implemented on the general-purpose I/O card.

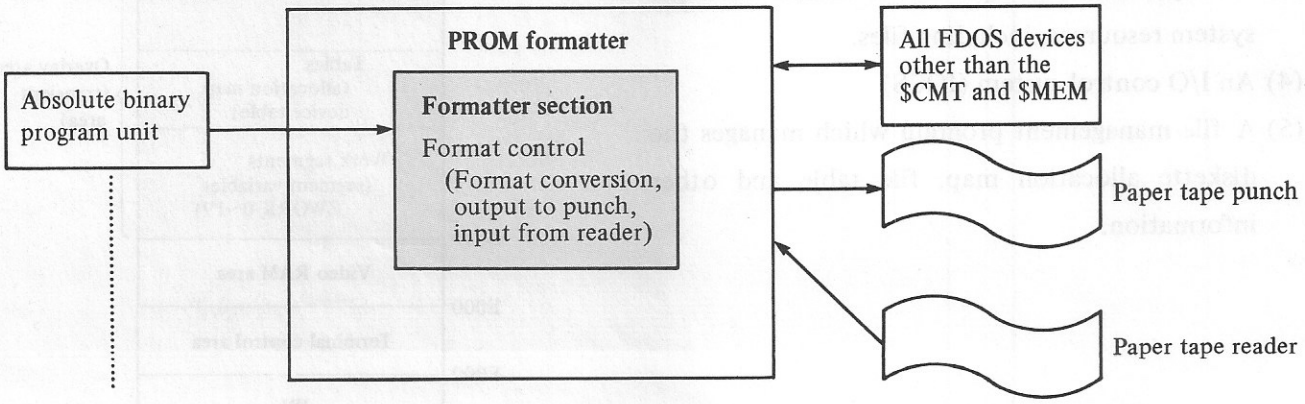


Fig. 2-14 PROM formatter configuration

3. FDOS ORGANIZATION

Figure 3-1 shows the files which are run under control of the SHARP MZ-80 series FDOS. The FDOS has the following features:

(1) Multistatement processing.

(2) Default argument processing.

(3) Allows wildcard characters in file references.

(4) File-oriented processing extended to I/O devices.

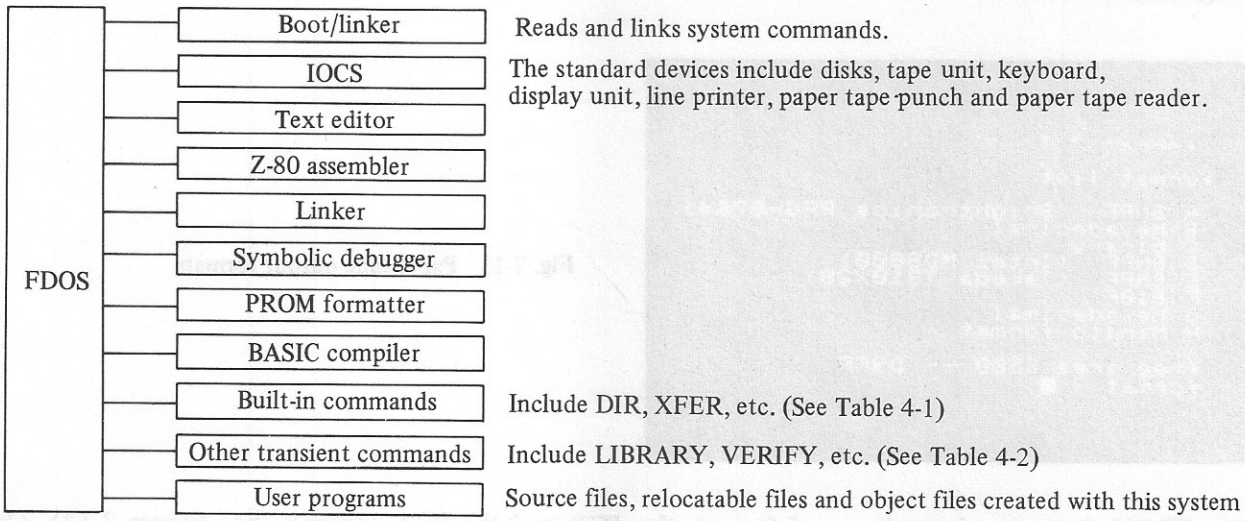


Fig. 3-1 FDOS file organization

Figure 3-2 shows the memory map for the above system resources. FDOS is made up of a resident section and an overlay section. Their resident section includes:

(1) A command line interpreter which interpretes and executes system commands.

(2) A boot linker which reads and links command files from the FDOS diskette.

(3) A supervisor call procedure which manages system resources, including files.

(4) An I/O control system (IOCS)

(5) A file management program which manages the diskette allocation map, file table and other information.

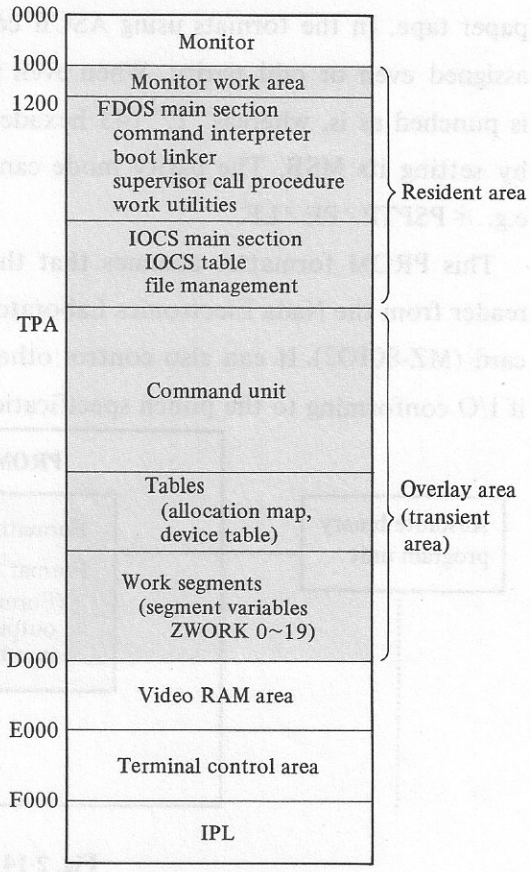


Fig. 3-2 FDOS memory map

3.1 Boot Linker

The FDOS transient commands (whose file mode is .SYS) are not resident in memory, but are stored in relocatable files on the system diskette. These programs exist not in absolute form but in relocatable form. When they are invoked, boot linker relocates them and specifies their loading addresses (see **Figure 3-3**).

These relocatable system files differ from relocatable files generated by the assembler in the way in which they are loaded into memory. The external symbol references of the system files have been resolved; these are just relocated by the boot linker. Accordingly, the control frame associated with each statement of the system programs contains only a field identifying the statement as having a relative address or absolute data and containing the byte count of the statement. When a relative address is indicated in the control frame, the system adds loading bias to the relative address to form an absolute address.

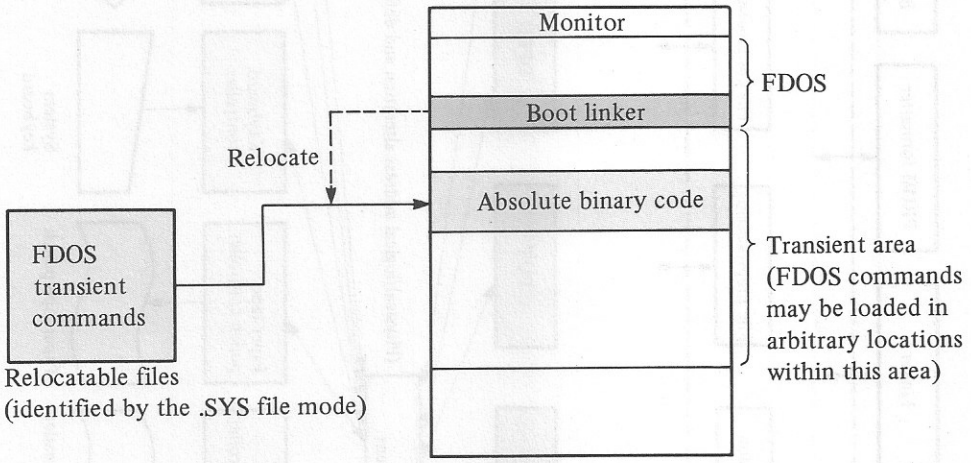


Fig. 3-3 Loading FDOS transient command with the FDOS boot linker

3.2 IOCS

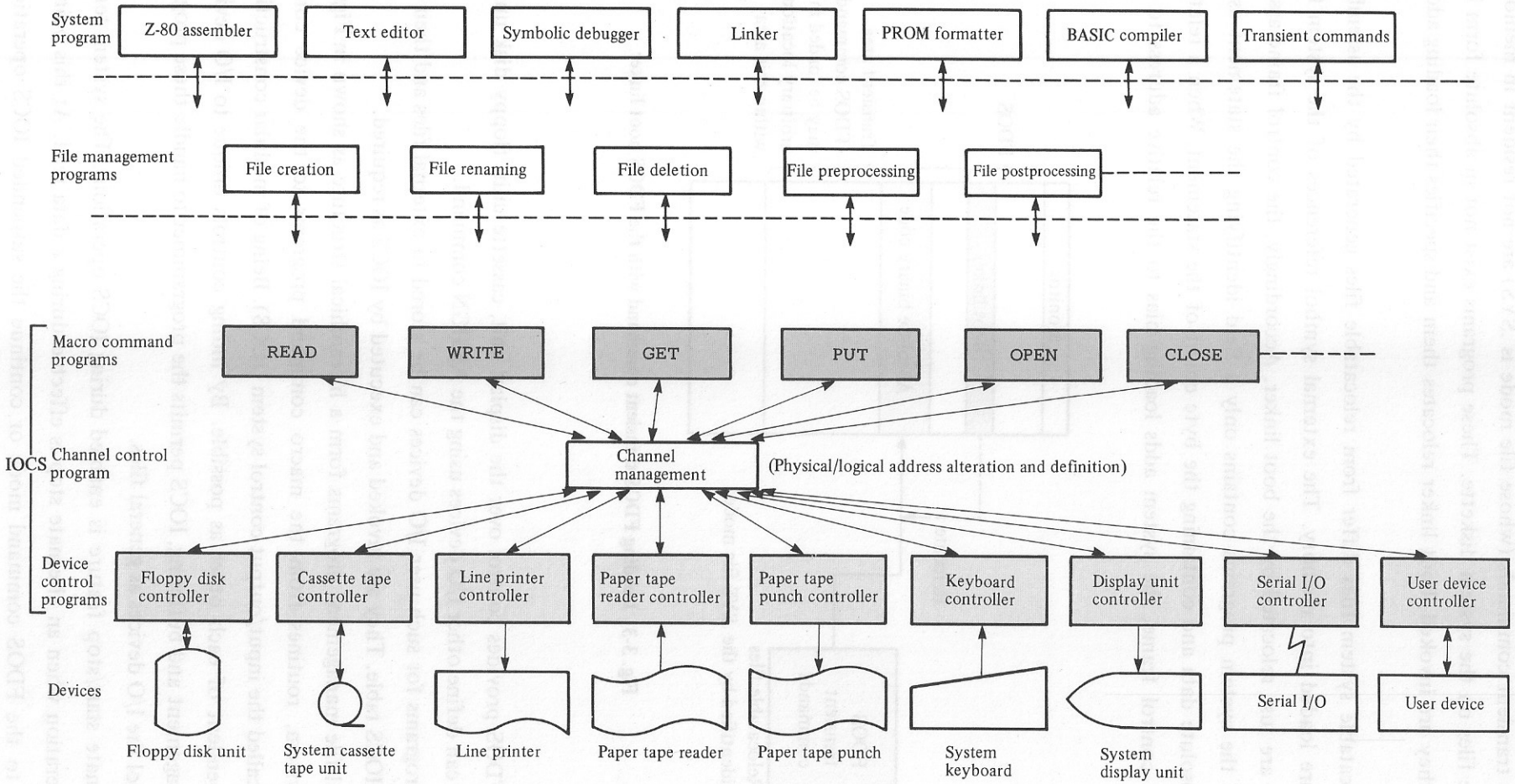
IOCS in FDOS provides control over the display unit, cassette unit, floppy disk unit and printer. The programmer can define other I/O devices using the ASSIGN command.

Control programs for such user I/O devices can be stored in external files and their names can be cataloged in the IOCS table. They are invoked and executed by IOCS as required.

The actual file management programs form a hierarchical structure as shown in **Figure 3-4**. In the MZ-80 series system, routines from the macro command programs to the device control programs are collectively called the input/output control system (IOCS). Being of modular construction, these programs are as independent of each other as possible. By hiding controls unique to I/O devices, such as device address management and buffering, IOCS permits the programmer to handle these programs as logical files and to control the I/O devices as general files.

The alternate start/stop feature is enabled during IOCS operations. The system temporarily suspends the read operation when an alternate stop is effected during a data read. At this point, the programmer can switch to the FDOS command mode or continue the suspended IOCS operation by effecting an alternate start.

Fig. 3-4 Hierarchical structure of file management programs



3.3 Dynamic Segmentation

Memory segmentation and relocation can be accomplished easily if a hardware relocation register is used. However, no presently available 8-bit microprocessor has such a register. Consequently, methods of simulating this function are commonly used. The boot linker previously mentioned can be thought of as a variation of such simulations. Here, a method of memory segmentation and assignment which leaves the memory image unchanged is described.

Two subroutines are used for memory segmentation as shown in **Figure 3-5** and **3-6.** These two subroutines and segment variables are maintained in fixed locations in the FDOS main program area. They are accessible to all programs. The 20 segment variables are initialized during preprocessing for each command and assigned values so that no memory segment exists. They are redefined as required during processing of each command, thus creating memory segments.

Fig. 3-5 Extending a specified segment

```
A ←2          ; Segment No. (0-19)
BC ←500       ; 500 bytes
CALL DOPEN    ; DYNAMIC OPEN
```

Segment No.	Segment variables	Results
0	ZWORK 0 : 5000	ZWORK 0 : 5000
1	ZWORK 1 : 5500	ZWORK 1 : 5500
2	ZWORK 2 : 6000+(500)	ZWORK 2 : 6500
3	ZWORK 3 : 6500+(500)	ZWORK 3 : 7000
4	ZWORK 4 : 7000+(500)	ZWORK 4 : 7500
5	ZWORK 5 : 7500+(500)	ZWORK 5 : 8000
6	ZWORK 6 : 8000+(500)	ZWORK 6 : 8500
7	ZWORK 7 : 8500+(500)	ZWORK 7 : 9000
⋮	⋮	⋮
18	ZWORK18 :29000+(500)	ZWORK18 : 29500
19	ZWORK19 :29500+(500)	ZWORK19 : 30000

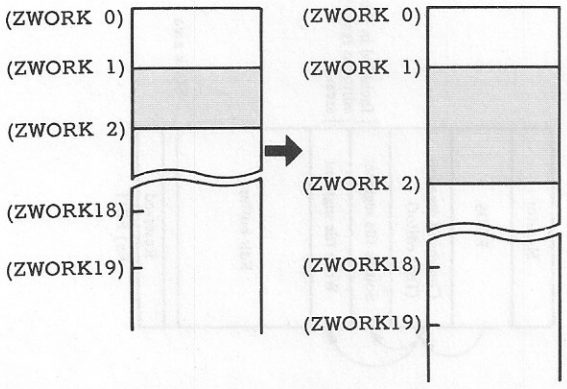


Fig. 3-6 Deleting a specified segment

```
A ←2          ; Segment No. (0-19)
BC ←500       ; 500 bytes
CALL DDELET   ; DYNAMIC DELETE
```

Segment No.	Segment variables	Results
0	ZWORK 0 : 5000	ZWORK 0 : 5000
1	ZWORK 1 : 5500	ZWORK 1 : 5500
2	ZWORK 2 : 6000–(500)	ZWORK 2 : 5500
3	ZWORK 3 : 6500–(500)	ZWORK 3 : 6000
4	ZWORK 4 : 7000–(500)	ZWORK 4 : 6500
5	ZWORK 5 : 7500–(500)	ZWORK 5 : 7000
6	ZWORK 6 : 8000–(500)	ZWORK 6 : 7500
7	ZWORK 7 : 8500–(500)	ZWORK 7 : 8000
⋮	⋮	⋮
18	ZWORK18 :29000–(500)	ZWORK18 : 28500
19	ZWORK19 :29500–(500)	ZWORK19 : 29000

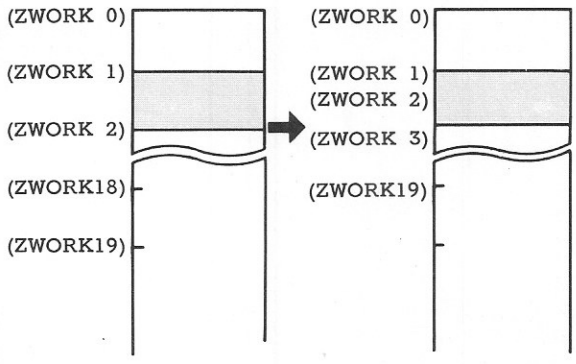
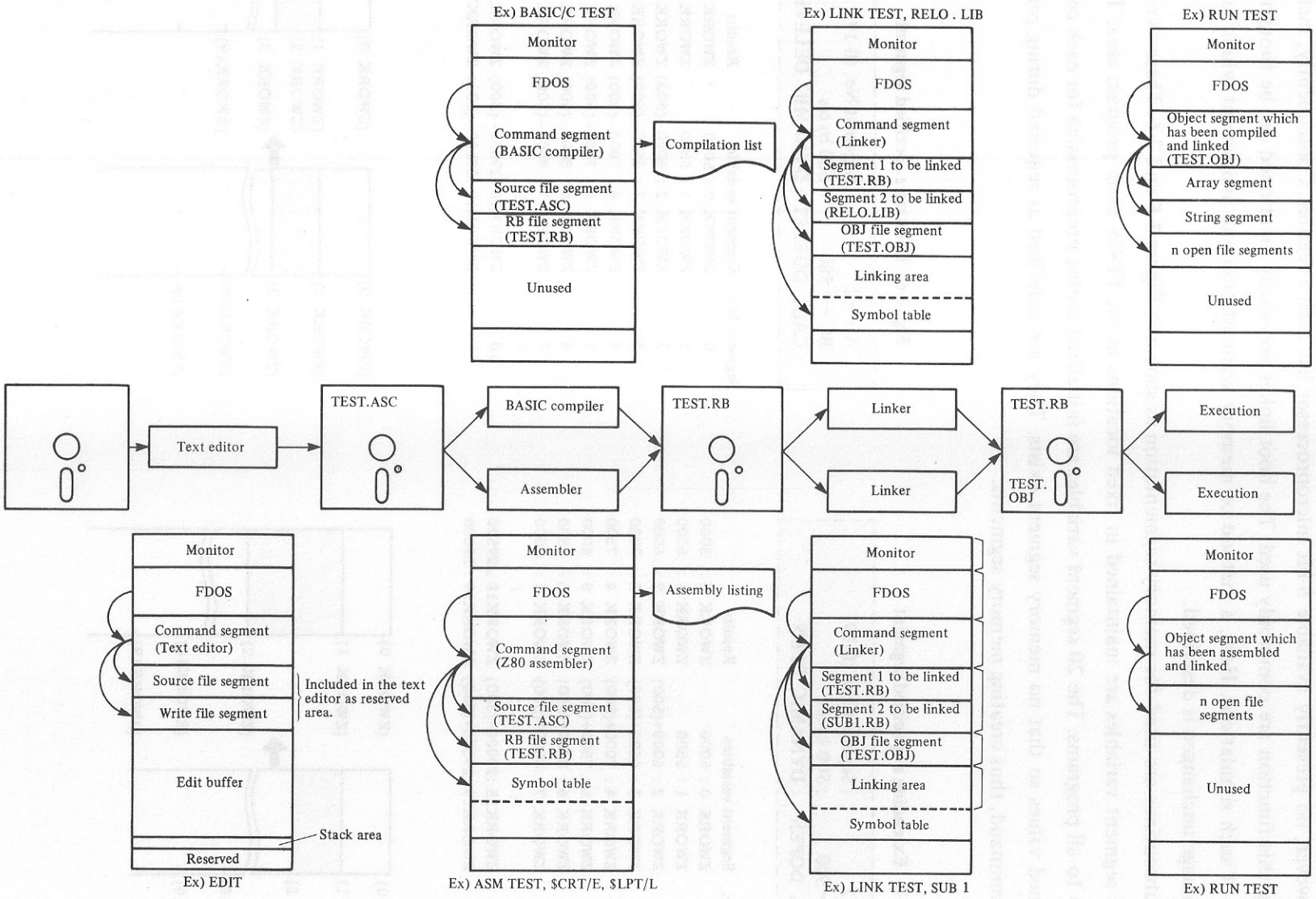


Fig. 3-7 Activation of Segments by FDOS



4. FDOS COMMAND USAGE

4.1 Program Development Under FDOS

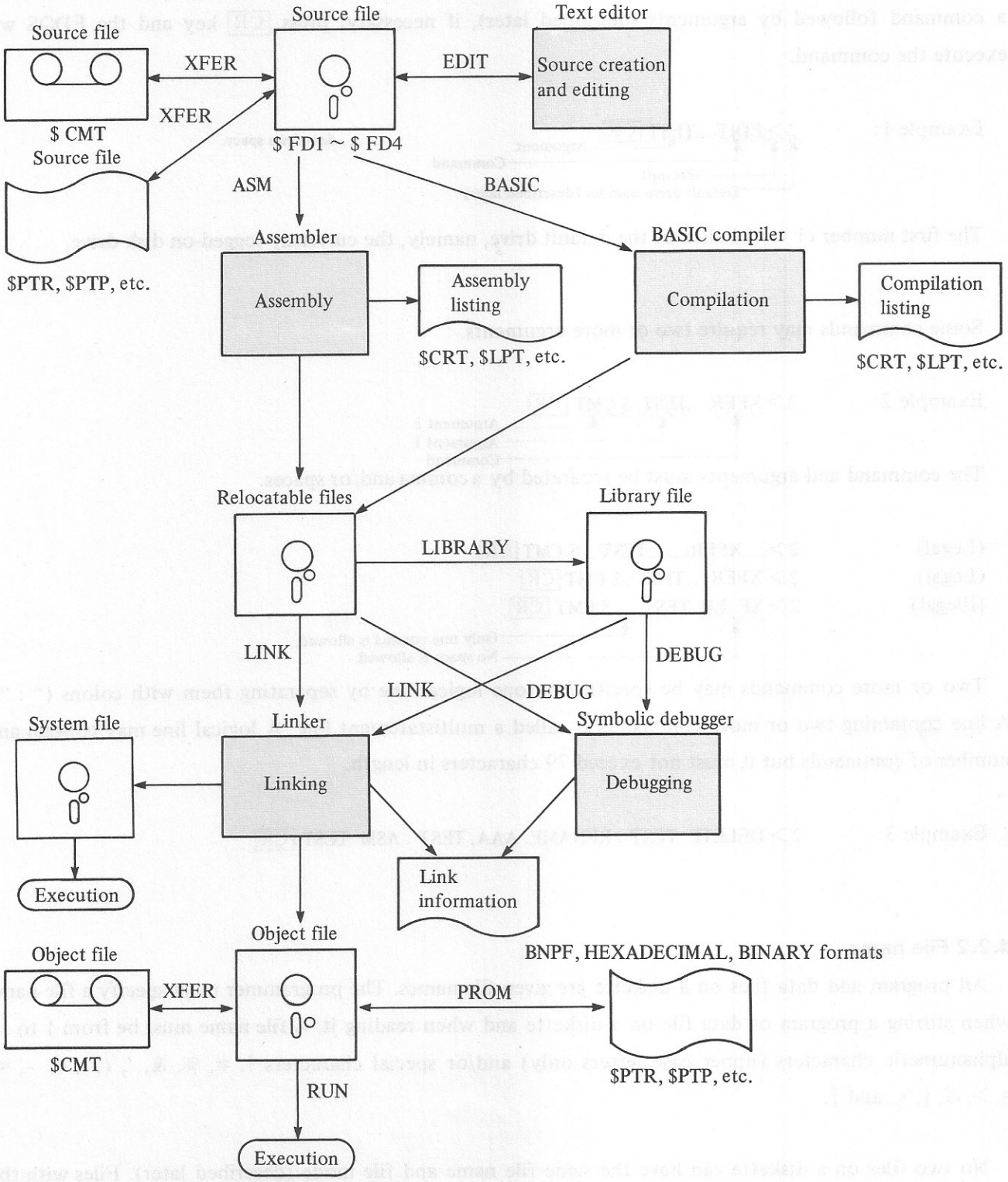


Fig. 4-1

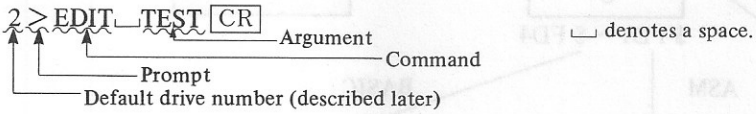
4.2 FDOS Command Coding Rules

This section describes the coding rules for FDOS commands.

4.2.1 Command line format

In the command mode, FDOS prompts for command entry with a number and the symbol ">". Enter a command followed by arguments (described later), if necessary, press [CR] key and the FDOS will execute the command.

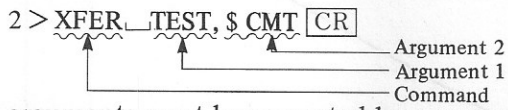
Example 1:



The first number (1 ~ 4) indicates the default drive, namely, the currently logged-on disk drive.

Some commands may require two or more arguments.

Example 2:

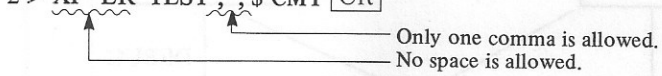


The command and arguments must be separated by a comma and/or spaces.

(Legal) 2 >␣XFER␣␣TEST␣$ CMT [CR]

(Legal) 2 > XFER , TEST , $ CMT [CR]

(Illegal)



Two or more commands may be specified on one logical line by separating them with colons (" : "). A line containing two or more commands is called a multistatement line. A logical line may contain any number of commands but it must not exceed 79 characters in length.

Example 3: 2 > DELETE TEST : RENAME AAA, TEST : ASM TEST [CR]

4.2.2 File name

All program and data files on a diskette are given file names. The programmer must specify a file name when storing a program or data file on a diskette and when reading it. A file name must be from 1 to 16 alphanumeric characters (upper case letters only) and/or special characters !, #, %, &, ', (,), +, –, <, =, >, @, [, \, and].

No two files on a diskette can have the same file name and file mode (described later). Files with the same file name may exist on a diskette if their file modes are different from one another.
(Files with the same file name and mode may exist on different diskettes).

4.2.3 File modes

The file mode identifies the kind of the file. It is usually used with a file name. The MZ-80 series file modes are listed below.

File mode

File mode	Meaning
. OBJ	Identifies an object file which contains Z80 machine code.
. ASC	Identifies a source file, such as one created by the text editor, which contains a stream of ASCII characters.
. RB	Identifies a relocatable file which contains pseudo-machine language code (relocatable binary code) generated by the assembler or compiler.
. LIB	Identifies a library file consisting of two or more relocatable files.
. SYS	Identifies a file containing a system program which runs under FDOS, such as the text editor and assembler.

4.2.4 File attributes

File attributes are information pertaining to file protection. There are four file attributes: 0, R, W and P. File attribute 0 indicates that a file is not protected. The other file attributes inhibit the use of specific commands as listed below.

File attribute	R	W	P
Inhibited FDOS Commands	TYPE XFER EDIT ASM LINK DEBUG PROM BASIC	DELETE RENAME	TYPE XFER EDIT ASM LINK DEBUG PROM BASIC DELETE RENAME
Inhibited BASIC Commands	ROPEN # INPUT #( )	PRINT #( )	ROPEN # INPUT #( ) PRINT #( )

4.2.5 File types

A file type indicates the file access method. There are two file types: sequential (S) and random (X). FDOS normally handles only sequential files. Random files can be accessed only by the DELETE, RENAME and CHATR commands. BASIC compiler is required to create, write to and read from random files.

4.2.6 Wildcard characters

The programmer can specify two or more files at a time by specifying wildcard characters in the file name and file mode. The wildcard characters " ? " and " ∗ " are used for file names and " .∗" is used for file modes.

Wildcard character " ? "

" ? " represents any one character. For example, assume that files ABC.ASC, ABC3.ASC, ABCD.RB, XYZ.ASC and ADCN.ASC exist on the currently logged-on disk. When the command.

TYPE A ? C ? . ASC

is entered, the contents of the files ABC3 . ASC and ADCN . ASC will be displayed.

Wildcard character " ∗"

" ∗ " Represents 0 or more characters.

A∗: Represents file names beginning with "A" e.g., A, A2, ABC

∗2 : Represents file names ending with "2" e.g., TEST2, SAMPLE2

P∗5: Represents file names beginning with "P" and ending with "5" e.g., PROGRAM5, PM5

Wildcard characters ".∗"

".∗" represents all file modes.

Examples:

DELETE PROG 1.∗	Deletes all files whose file name is PROG1
XFER ∗ .ASC, $ PTP	Punches all files whose file mode is .ASC.
DIR A ∗B∗ ? 3 .RB	
DELETE ∗ . ∗	Deletes all files on the diskette.

4.2.7 Drive number and volume number

A drive number refers to the drive number of a floppy disk drive (MZ-80FB or MZ-80FBK). Drive numbers 1 through 4 are assigned device names $FD1 through $FD4 respectively.

A volume number (1-127) is a number identifying a diskette. FDOS checks this number for validity each time it accesses a file.

4.2.8 Basic device name

FDOS can handle the following I/O devices:

$KB : MZ-80A system keyboard

$CRT : MZ-80A system display unit

$FD1 :
$FD2 :
$FD3 :
$FD4 : } Floppy disk drives (MZ-80FB or MZ-80FBK)

$CMT : System cassette unit

$LPT : Optional printer

$MEM : A part of main memory regarded as an I/O resource.

The system automatically reserves an unused area as $MEM. This area is released by the DELETE $MEM command or when an error occurs.

4.2.9 Auxiliary device name

Auxiliary devices are devices whose control programs are not resident in the FDOS area in memory. Their control programs are stored in external files. An auxiliary device name is assigned to an auxiliary device control program using the ASSIGN command to allow IOCS to manage the control program.

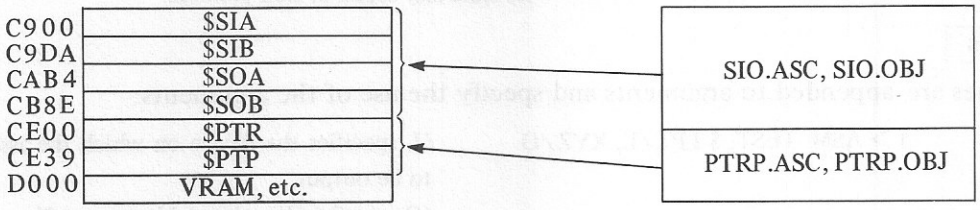
$PTR :
$PTP :
} Paper tape reader and punch. The user must prepare an interface circuit for these using a universal interface card. The system contains their control programs, however. For details, refer to "PAPER TAPE PUNCH AND READER INTERFACE" in the Appendix.

$SIA : Serial input port A
$SIB : Serial input port B
$SOA : Serial output port A
$SOB : Serial output port B
} The interface card for these I/O ports is optional.

$USR1 :
$USR2 :
$USR3 :
$USR4 :
} These device names are provided for user-supplied I/O devices. The control program must be supplied by the user.

To use these device names, prepare a machine language area using the LIMIT command, load the corresponding auxiliary device control program into the area using a LOAD command and link the program with the I/O controller of FDOS using an ASSIGN command. The auxiliary device control programs are supplied in the form of object files and ASCII files. In general, use the object files. If you want to change the loading address, assemble and link the ASCII files with FDOSEQU.LIB from the master diskette.

The loading address of each auxiliary device control program is shown below.



1. Memory map　　2. Control programs

Example 5:
```
1 > LIMIT $C900
1 > LOAD SIO  PTRP
1 > ASSIGN   $SIA  $C900  $SIB  $C9DA  $SOA  $CAB4  $SOB  $CB8E
             $PTR  $CE00  $PTP  $CE39
```

Example 6: EXEC $FD1 ; LOADAUX

All the auxiliary device control programs are loaded since file LOADAUX.ASC contains the above commands.

Notes:

1. Any file input from the keyboard ($KB) is terminated by pressing the BREAK key. For example, execution of the command

   1 > XFER $KB, XYZ

   is terminated when the programmer presses the BREAK key.

2. The end of files from $PTR is identified by the null code (00H) following the data area (null codes in the feed area are ignored).
3. $CMT and $MEM can be accessed only by the built-in commands and programs compiled by the BASIC compiler. When they are used by other programs, the error message

   no usable device

   is issued.
4. $CMT can handle only .ASC and .OBJ mode files. $KB, $CRT, $LPT, $PTR, $PTP and $MEM can handle only .ASC mode files (error message "il file mode" is issued if an illegal file mode file is used with one of these devices).
5. $PTP and $PTR automatically skip the tape feed portions.

4.2.10 Switches

Switches follow command names or arguments and specify optional command functions. There are three types of switches.

Global switches

Global switches are appended to command names and specify the mode in which the command is to be executed. Two or more switches may be specified for a command as shown in Example 8. In such cases they may be placed in any order.

Example 7: `1 > DATE/P` — /P denotes LPT.

Example 8: `1 > LINK/P/T TEST` — /P denotes LPT.

/T denotes the symbol table.

Invalid: `1 > LINK␣/P␣/␣T␣TEST`

Local switches

Local switches are appended to arguments and specify the use of the arguments.

Example 9: `1 > ASM TEST, $ LPT/L, XYZ/O`

/L specifies the device on which the assembly listing is to be output.
/O specifies the relocatable output file.

Device switches

Device switches are appended to device names. Their format is identical to that of local switches. The legal device switches are /PE, /PO, /PN and /LF. These switches can be appended only to devices $PTR, $PTP, $USR1 ~ 4.

The meanings of the device switches are listed below.

Switch	Input	Output
/PE	Specifies that data is to be checked for even parity.	Specifies that even partiy is to be used. (Note)
/PO	Specifies that data is to be checked for odd parity.	Specifies that odd parity is to be used. (Note)
/PN	Specifies that bit 7 (MSB) of input data is to be set to 0.	(Note)
/LF	Invalid	Specifies that CR is to be followed by LF.

Note: An error is generated (il data) if the MSB of the data is set to 1 from the beginning (e.g., graphic characters).

Note:

Any switch following the first argument of the RUN command is treated as a global switch.

Example 10: 1 > RUN␣ASM48/P␣TEST, XYZ/O

Local switch
Global switch

The meanings of the individual global switches are described in the related command descriptions.

4.2.11 Default assumptions

The general format of a file specification (valid for $FD1–$FD4 and $CMT) is given below.

Example 11: $ FD2 ; PROG2 . ASC $ CMT ; TEST2 . OBJ

File mode
File name
Device name
File mode
File name
Device name

The programmer can omit portions of the complete file specification as explained below.

Default drive

The device name may be omitted as exemplified below.

Example 12: 2 > LINK TEST1, $FD3 ; TEST2, TEST3

In the above example, the system assumes the name of the currently logged-on disk drive (identified by "2>") before TEST1 and TEST3. Consequently, the above command line is equivalent to the following:

2 > LINK $FD2 ; TEST1, $FD3; TEST2, $FD2; TEST3

The default drive can be changed by:

1. Executing the DIR command or
2. Moving the cursor to the left of the prompt ">" and changing the drive number (e.g., changing "2>" to "1>").

Default file name

The file name may be omitted when reading files from the cassette tape unit ($CMT). When a file name is omitted in the XFER command or other similar command (See example 13), the system assumes an appropriate file name.

Example 13 : XFER $ FD1 ; ABC . ASC, $ FD2

The system assumes $FD2; ABC. ASC.

Default file mode

When the file mode is omitted, the system makes an appropriate default assumption according to the command. See the individual command descriptions.

Notes:

1. Both device name and file name cannot be omitted simultaneously.
2. No file name can be assigned to devices other than $FD1 through $FD4 and $CMT.

4.2.12 Arguments

There are several argument formats.

1. Device name + File name + File mode

   Examples : $ FD1 ; ABC . ASC $ CMT ; XYZ . OBJ $ FD2 ; ＊ . ＊

2. Device name + File name. The file mode is omitted.

   Examples : $ FD1 ; ABC $ FD2 ; A＊ $ CMT ; TEST

3. File name + File mode. The device name is omitted (default drive).

   Examples : TEST3 . RB ＊. ASC PROG? . RB

4. Device name

   a. When the file name and mode are omitted or when the device name proper is to be specified.

      Examples : $ FD1 $ CMT

   b. When neither file name nor mode can be specified.

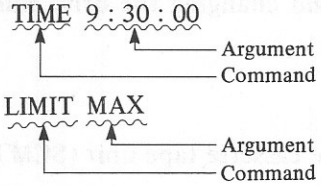
      Examples : $ PTR $ CRT $ LPT

5. Hexadecimal constant

   Examples : $ 1200 $ C000

6. Special arguments

   Examples :



4.3 Using FDOS Commands

4.3.1 ASM

Transient

Format

ASM filename

Function

The ASM command assembles the source program in the source file specified by the argument, outputs the result to a relocatable file and outputs an assembly listing to the specified file or device.

Default file mode

.RB when local switch /O is specified; otherwise, .ASC.

Switches

Global switches:

None: A relocatable file is generated.

/N: No relocatable file is generated.

Local switches:

None: Specifies that the specified source file is to be assembled.

/O: Specifies that the relocatable code is to be output to a file under the selected name.

/E: Specifies that only error statements are to be output to the selected file or device.

/L: Specifies that the assembly listing is to be output to the selected file or device.

Wildcard characters

Not allowed

Examples

(1) ASM TEST

Assembles source file TEST.ASC and generates relocatable file TEST.RB.

(2) ASM TEST, $ LPT/L, XYZ/O

Assembles source file TEST.ASC, generates relocatable file XYZ.RB and outputs the assembly listing to LPT.

(3) ASM/N TEST, $CRT/E, $ SOA/L

Assembles source file TEST.ASC while displaying error statements (including external symbol references) and outputting the assembly listing to SOA. No relocatable file is generated.

(4) ASM TEST, $ FD2 ; TEST1/L, $FD2 ; TEST1 .RB/O

Assembles source file TEST.ASC and saves relocatable file TEST1.RB and assembly listing TEST1.ASC on FD2.

(5) ASM TEST, $ LPT/L, $ 2000

Assembles source file TEST.ASC, generates relocatable file TEST.RB and outputs the assembly listing to LPT with a bias of 2000H added.

4.3.2 ASSIGN

Transient

Format

ASSIGN devicename1, $nnnn, ..., devicenameN, $nnnn

Function

The ASSIGN command assigns logical device names to user-supplied I/O control routines.

Switches

None.

Wildcard characters

Not allowed.

Examples

(1) **LIMIT $C000**
ASSIGN $USR1, $C000
Assigns device name $USR1 to the user I/O control routine at address $C000.

(2) **ASSIGN $USR2, $C200, $USR3, $C400**
Assigns $USR2 to the routine at address $C200 and $USR3 to the routine at address $C400.

(3) **ASSIGN $PTP, $C600**
Assigns $PTP to the new PTP routine at address $C600 in place of the PTP control routine in FDOS.

Programming notes

(1) When a device name is assigned more than once, the last assignment is taken.

(2) To cancel an assignment, set the address operand to $FFFF.
Example : ASSIGN $USR1, $FFFF This command cancels $USR1.

(3) When an I/O control routine is destroyed by execution of a new LIMIT or LOAD command it is necessary to cancel the device assignment for that routine using the above procedure.

4.3.3 BASIC

Transient

Format

BASIC filename

Function

The BASIC command compiles the source program written in BASIC language identified by the argument and outputs the BASIC listing.

Default file mode

.RB when local switch /O is specified; .ABC otherwise.

Switches

Global switches

/N: Specifies that no relocatable file is to be generated.

/C: Specifies that the BASIC listing is to be displayed on CRT.

/P: Specifies that the BASIC listing is to be printed on LPT.

(Note that switches /C and /P cannot be specified simultaneously.

Local switches

None: Specifies that the specified source file is to be compiled.

/O: Specifies that the relocatable file is to be output to the selected file.

Wildcard characters

Not allowed.

Examples

(1) **BASIC TEST**

Compiles source file TEST.ASC and generates relocatable file TEST.RB.

(2) **BASIC/C TEST, XYZ/O**

Compiles source file TEST.ASC, generates relocatable file XYZ.RB and displays the BASIC listing on CRT.

(3) **BASIC/N/P TEST**

Compiles source file TEST.ASC and prints the BASIC listing on LPT. No relocatable file is generated.

Programming notes

The compiler terminates generation of the relocatable file when it detects an error during compilation.

4.3.4 BOOT

Built-in

Format

BOOT

Function

Terminates execution of FDOS and activates the MZ-80A system IPL (Initial Program Loader). (same as monitor F command)

Programming notes

The system program is loaded into memory when IPL is activated. Therefore, former memory contents are cleared.

4.3.5 CHATR

Built-in

Format

CHATR sign, filename1, attribute1, ..., filenameN, attributeN

Function

The CHATR command changes the attributes of a specified file.

Default file mode

.ASC

Switches

None.

Wildcard characters

Not allowed.

File attributes

0 : None.

R : Read-protected file

W : Write-protected file

P : Permanent file

Examples

(1) **CHATR KEY, TEST, R**

Assigns the password "KEY" to file TEST.ASC and declares the file as a read-protected file.

(2) **CHATR SECRET, TEST.OBJ, 0**

Deletes the file attributes of file TEST.OBJ. The specified password, "SECRET", is matches with the password specified for the file before the command is actually executed.

(3) **CHATR**

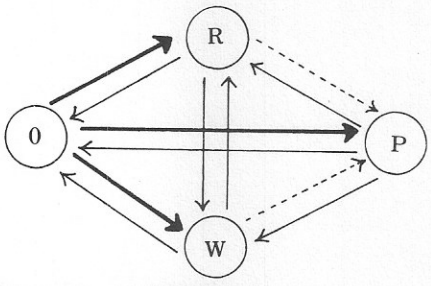
Allows the programmer to interactively specify the sign, file name and attribute in that order.

(4) **CHATR sign**

Allows the programmer to interactively specify the file name and attribute in that order.

Programming note

The interrelationship of the file attributes is shown below.



Set sign.

Check sign.

Does not check sign.

4.3.6 CONVERT

Transient

Format

CONVERT

Function

Converts a file generated with the SA-5000 series BASIC interpreter or the D-BASIC SA-6000 series into a file usable under FDOS, or converts a file generated with FDOS into a file usable under the

SA-5000 series or SA-6000 series. The relationship between file modes handled by this command is as follows.

BASIC		FDOS
BTX	⟷	ASC
BSD	⟷	ASC
OBJ	⟷	OBJ

Default file mode

None.

Switches

None.

Wildcard characters

Not allowed.

Example

```
2 > CONVERT
Choose one from:
  1 : BTX → ASC
  2 : ASC → BTX
  3 : BSD → ASC
  4 : ASC → BSD
  5 : OBJ → OBJ
(1 ~ 5) ?
Source drive No. (1 ~ 4, CMT = 0) ?2
Source file name ? SAMPLE
Destination drive No. (1 ~ 4, CMT = 0) ?3
Destination file name ? SAMPLE
End of convert
```

Enter 1 ~ 4 for the $FD and 0 for the $CMT.

Programming notes

(1) Never intermix D-BASIC format diskettes and FDOS format diskettes. Otherwise, disk contents may be destroyed.

(2) Since the syntax of D-BASIC and that of the BASIC compiler differ slightly, there are some cases in which programs converted with the CONVERT command cannot be compiled by the BASIC compiler without some modification. Use the text editor to modify such programs before compiling them with the BASIC compiler.

(3) A BRD file cannot be converted. First convert it into a BSD file, then execute the CONVERT command.

Refer page 60 for further information.

4.3.7 COPY

Transient

Format

COPY

Function

The COPY command copies the contents of the source diskette to the destination diskette. The programmer can specify only predetermined types of diskettes as the destination and source diskettes as summarized in the table below.

Source	Destination	Allowed/disallowed	Remarks
(Any diskette)	Master	Disallowed	
Master	Submaster	Allowed	
Master	Slave	Allowed	The destination diskette becomes a submaster diskette.
Submaster	Submaster	Disallowed	
Submaster	Slave	Disallowed	
Slave	Submaster	Allowed	The destination diskette becomes a slave diskette.
Slave	Slave	Allowed	

It is desirable to create a submaster diskette from the master diskette using the COPY command and to use this submaster diskette during normal operation. It is also desirable to make copies at appropriate times when the original diskette is updated to prevent errors due to physical defects in the disk or software errors or inadvertent use of the DELETE command.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

(1) FDOS always copies from \$FD1 to \$FD2 when the system has two or more floppy disk units.

```
2 > COPY
Destination diskette's sign ?BACKUP     ← Proceeds to the next step if the passwords match.
Insert source into $FD1                 ← Insert the source diskette in drive FD1.
Destination into $FD2, ⇩ Space key      ← Insert the destination diskette in drive FD2, then press the [SP] key.
2 >                                       Copying is completed.
```

4.3.8 DATE

Built-in

Format

DATE mm . dd . yy

Function

The DATE command sets or displays the system calender date in the month.date.year format. This information is assigned to each file when it is saved on a diskette. The date is not automatically updated, however.

Default file mode

None.

Switches

Global switch/P: Specifies that the date is to be printed on LPT.

Wildcard characters

Not allowed.

Examples

(1) **DATE 1 . 1. 82**

Sets the system calender date to Janualy 1st, 1982

(2) **DATE**

Displays the current date on CRT.

(3) **DATE/P**

Prints the current date on LPT.

4.3.9 DEBUG

Transient

Format

DEBUG filename1, ..., filenameN

Function

The DEBUG command links and loads relocatable files specified by the arguments to form an object program in memory for debugging.

Default file mode

.OBJ when local switch/O is specified; .RB otherwise.

Switches

Global switches

None: Specifies that only the link information is to be displayed on CRT.

/T: Specifies that the symbol table information is to be output (on CRT unless global switch /P is specified).

/P: Specifies that the link and symbol table information is to be printed on LPT when global switch /T is specified.

Local switch

/O: Specifies that the object file is to be created under the selected file name.

Wildcard characters

Not allowed.

Examples

(1) **DEBUG TEST1, TEST2**

Links and loads relocatable files TEST1.RB and TEST2.RB and waits for a debugger command. The link information is displayed on CRT.

(2) **DEBUG/T/P TEST, TEST/O**

Loads relocatable file TEST.RB, prints the link and symbol table information on LPT and generates object file TEST.OBJ.

(3) **DEBUG TEST1, $1000, TEST2, TBL $20**

Links and loads relocatable files TEST1.RB and TEST2.RB and reserves $1000 bytes of free area in memory between them. The symbol table size is set to $2000 (approximately 8K bytes). When the table size is not specified, the debugger automatically allocates 6K bytes for it.

(4) **DEBUG**

Invokes the symbolic debugger and enters the command mode.

4.3.10 DELETE

Built-in

Format

DELETE filename1, ..., filenameN

Function

The DELETE command deletes the files specified by the arguments except those with the W or P file attribute.

Default file mode

.ASC

Switches

Global switches /C : When this switch is specified, the system displays each file on CRT for confirmation. The file is deleted when the programmer presses the [Y] key and skipped when he presses the [N] key.

/N : Specifies that no deleted file is to be displayed. (The programmer must not specify /N and /C simultaneously.)

Wildcard characters

Allowed.

Examples

(1) **DELETE TEST . ＊**

Deletes all files whose file name is TEST.

(2) **DELETE /C ＊ .OBJ**

Displays all files with a file mode of .OBJ on CRT for confirmation before deleting them.

(3) **DELETE $FD2 ; ＊.＊**

Deletes all files on FD2 except those with the file attribute P or W. To delete file-protected file, it is necessary to cancel the file protect attributes with the CHATR command.

(4) **DELETE $ MEM**

Deletes file $ MEM.

4.3.11 DIR

Built-in

Format

DIR devicename (filename)

Function

Displays the contents of the directory specified by devicename of filename. "devicename" must refer to a floppy disk unit.

Default file mode

. ＊

Switches

Global switch /P : Specifies that the directory is to be printed on LPT.

Wildcard characters

Allowed.

Examples

(1) **DIR $FD2**

Displays the file information of all files on the diskette in FD2 on CRT. FD2 is designated as the default drive.

(2) **DIR/P**

Prints the file information of all files on the diskette in the current default drive on LPT. The directory device remains unchanged.

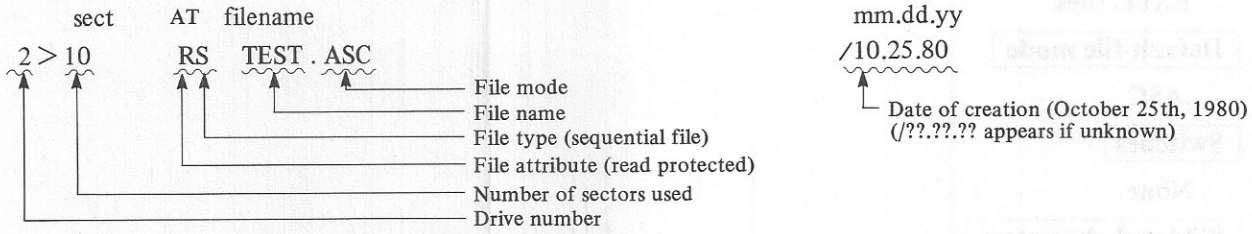
(3) **DIR TEST**

Displays on CRT the file information of all files on the diskette in the current default drive whose file name is TEST.

(4) **DIR $FD2 ; ＊ . ASC**

Displays the file information of all source files on the diskette in FD2 on CRT. FD2 is designated as the default drive.

Programming notes



4.3.12 EDIT

Transient

Format

EDIT filename

Function

The EDIT command invokes the text editor to create a new source file or edit an existing source file.

Default file mode

.ASC

Switches

None.

Wildcard characters

Not allowed.

Examples

(1) **EDIT**

Invokes the text editor and enters the command mode.

(2) **EDIT TEST**

Invokes the text editor, reads source file TEST. ASC and enters the command mode.

(3) **EDIT $FD2 ; TEST**

Invokes the text editor, reads source file TEST. ASC from the $FD2 and enters the command mode.

4.3.13 EXEC

Built-in

Format

EXEC filename

Function

The EXEC command executes the contents of the file specified by the argument as FDOS commands. A device name may be specified in place of filename. Files containing FDOS command are called EXEC files.

Default file mode

.ASC

Switches

None.

Wildcard characters

Not allowed.

Examples

(1) **EXEC MACRO**

Executes the contents of source file MACRO.ASC assuming that the file consists of FDOS commands. When the file MACRO.ASC contains the command lines shown below, the system executes the commands in sequence from the top to the bottom.

```
ASM $FD2 ; TEST
LINK /T/P $FD2 ; TEST
CHATR KEY, $FD2 ; TEST.OBJ, W
RUN $FD2 ; TEST
```

```
3 > FREE
DIR /P $FD2
```

(2) **EXEC MYDEVICE**

Sequentially executes the command lines contained in source file MYDEVICE.

```
LIMIT $C000                          ← Limit the FDOS area to $C000.
LOAD MYPRINTER                       ← Set the loading and execution addresses to $C000.
LOAD MYLIGHTPEN                      ← Set the loading and execution addresses to $C800.
ASSIGN $USR1, $C000, $USR2, $C800    ← Assign user I/O names to user programs.
```

(3) **EXEC ABC**

Executes the routine in file DEF repeatedly if file ABC.ASC contains the following routine.

```
RUN DEF
EXEC ABC
```

Programming notes

(1) Since the EXEC command executes the commands specified in a file as macro commands, it cannot be specified on a multistatement line as shown below.

```
EXEC MACRO : TYPE MACRO
```

(2) The specified file may have the file attribute R, W or P. However, execution of files with the attribute R or P is not displayed.

(3) When an error occurs during execution of an EXEC file, the system immediately terminates processing and waits for entry of a new FDOS command from the keyboard.

(4) When the file name START-UP is assigned to an EXEC file, that file will be automatically executed when FDOS is activated.

4.3.14 FORMAT

Transient

Format

FORMAT $FDn

Function

The FORMAT command formats (initializes) a new diskette.

The user must always format new diskettes before using them.

Default file mode

None.

Switches

None.

Wildcard characters

Not allowed.

Examples

(1) **FORMAT $FD2**

```
FDOS diskette formatting
Insert diskette into $FD2, ⇩ space key
New sign ? SHARP
Volume No. ? 50
END
Insert diskette into $FD2, ⇩ space key
Break ←Press the [BREAK] Key to return to FDOS.
```

The above **interaction shows an example of formatting a completely new diskette.** "sign" prompts for a password to be given at the diskette. When this diskette is resubmitted for formatting, the system checks for a password match before actually reformatting the diskette. "Volume No." prompts for a volume number to be assigned to the diskette. The programmer can specify any number from 1 to 127. The volume number should be unique.

(2) **FORMAT**

```
FDOS diskette formatting
Insert diskette into $FD1,      ⇩ space key
Old sign ? SHARP ← The system matches the password entered with that stored on the diskette and proceeds to
                   the next step if they match.
New sign ? MZ-80 ← Set a new password.
Volume No. ? 100
END
Insert diskette into $FD1,      ⇩ space key
Break ← Press the [BREAK] key to return to FDOS.
```

The above **interaction shows an example of reformatting a previously formatted diskette.** The meanings of "sign" and "Volume No." are identical to those in example (1).

Programming notes

The following message will be displayed if a diskette cannot be initialized because of defects, etc.

(1) **bad track #nn**

When this message is displayed, the XFER command can be executed for the diskette but the COPY command cannot.

(2) **no usable diskette**

When this message is displayed, this diskette is not usable.

4.3.15 FREE

Built-in

Format

FREE $FDn

Function

The FREE command displays the number of used sectors, the number of unused sectors, and/or the volume number of the diskette in the specified floppy disk unit.

Default file mode

None.

Switches

Global/P : Specifies that the disk usage information is to be printed on LPT.

Wildcard characters

Not allowed.

Examples

(1) FREE $FD2

$ FD2 vol : 100 left : 1072 used : 48

(2) FREE/P

Prints the same information as given in example (1) on LPT, except that the information pertains to the diskette in the default drive.

Programming note

A diskette is comprised of 1120 sectors (each consisting of 256 bytes). Of these 1120 sectors, however, 48 sectors are reserved by the system as FDOS areas. Consequently, used:48 is indicated for new diskettes.

4.3.16 HCOPY

Built-in

Format

HCOPY message

Function

HCOPY prints the contents of the CRT screen from the upper left position to the current cursor position on LPT as is with a message.

Default file mode

None.

Switches

None.

Wildcard characters

Not allowed.

Examples

(1) HCOPY

Prints a copy of the CRT screen on LPT.

(2) HCOPY ■ SHARP-FDOS

Prints a copy of the CRT screen on LPT after outputting a form feed and the specified message.

Programming note

(1) Characters which can be used for messages are ASCII codes 00H-7FH, except for "/" and ":".

(2) The following are LPT mode control codes.

- H Paging: Feeds the paper to the position where power has been turned on.
- J Suppressed spacing: Used for graphic display, etc.
- K Double size characters: Used for titles, etc.
- C Clear: Clears the J and K functions.

4.3.17 LIBRARY

Transient

Format

LIBRARY filename1, ..., filenameN

Function

The LIBRARY command reads the relocatable files specified by the arguments to form a library file.

Default file mode

.LIB when local switch /O is specified; .RB otherwise.

Switches

Global switches

None: Link information pertaining to the relocatable files is displayed on CRT.

/P: Specifies that the link information is to be printed on LPT.

Local switches

None: The first filename specified is used as the name of the library file.

/O: Specifies that the library file is to be created with the selected file name.

Wildcard characters

Not allowed.

Examples

(1) **LIBRARY TEST1, TEST2**

Reads relocatable files TEST1.RB and TEST2.RB to generate library file TEST1. LIB. The link information is displayed on CRT.

(2) **LIBRARY/P TEST1.LIB, TEST2, XYZ/O**

Reads relocatable files TEST1.LIB and TEST2.RB and generates a library file named XYZ.LIB. The link information is printed on LPT.

4.3.18 LIMIT

Transient

Format

LIMIT $nnnn

Function

The LIMIT command sets the FDOS area boundary at address $nnnn.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

(1) **LIMIT $C000**

Limits the FDOS area to $C000 and frees the higher area.

(2) **LIMIT MAX**

Sets the FDOS area to the maximum available address.

Programming note

The LIMIT command cannot be specified in a multistatement as shown below.

Illegal: LIMIT $B000 : DIR $FD2

4.3.19 LINK

Transient

Format

LINK filename1, ..., filenameN

Function

The LINK command links the relocatable files specified by the arguments to generate an object or system file.

Default file mode

.OBJ when local switch /O is specified; .RB otherwise.

Switches

Global switches

None: Only the link information is displayed on CRT.

/T: Specifies that the symbol table is to be output (on CRT unless global switch /P is specified).

/P: Specifies that the link and symbol table information is to be output to LPT (when global switch /T is specified).

/S: Specifies that a system file is to be generated.

Local switches

None: The first filename specified is used as the name of the object file.

/O: Specifies that the object file is to be created under the specified file name. If global switch /S is specified, specifies that the system file is to be created under the specified file name.

Wildcard characters

Not allowed.

Examples

(1) **LINK TEST1, TEST2**

Links relocatable files TEST1.RB and TEST2.RB and generates an object file named TEST1.OBJ. The loading and execution addresses of the object file are automatically set to the beginning address managed by FDOS. The link information is displayed on CRT.

(2) **LINK/T/P TEST1, TEST2, XYZ /O**

Links relocatable files TEST1.RB and TEST2.RB and generates object file XYZ.OBJ. The loading

and execution addresses of the object file are set to the beginning address managed by FDOS. The link and symbol table information is output to LPT.

(3) **LINK $C000, TEST, FDOSEQU.LIB, EXEC$C100**

Links TEST.RB and FDOSEQU.LIB and generates object file TEST.OBJ, specifying $C000 as the loading address. The execution address of the object file is $C100.

(4) **LINK TEST1, $1000, TEST2, TBL $20**

Links file TEST1.RB (specifying the beginning of the FDOS area as the loading address), then links and loads file TEST2.RB, reserving $1000 bytes of free area between the two files. The symbol table size is set to 8K ($2000) bytes.

4.3.20 LOAD

Transient

Format

LOAD filename1, ..., filenameN

Function

The LOAD command loads the object files specified by the arguments in areas outside the area managed by FDOS.

Default file mode

.OBJ

Switches

None.

Wildcard characters

None.

Example

(1) **LOAD TEST1, TEST2**

Loads object files TEST1.OBJ and TEST2.OBJ into memory areas outside the area managed by FDOS. The programmer must create object files so that they are to be loaded in appropriate addresses.

4.3.21 MLINK

Transient

Format

MLINK filename1, ..., filenameN

Function

The MLINK command links the relocatable files specified by the arguments to generate an object file.

Default file mode

.OBJ when local switch /O is specified; .RB otherwise.

Switches

Global switches

None: Only the link information is displayed on the CRT.

/T: Specifies that the symbol table is to be output (on the CRT unless global switch /P is specified).

/P: Specifies that the link and symbol table information is to be output to the LPT (when global switch /T is specified).

Local switches

None: The first file name specified is used as the name of the object file.

/O: Specifies that the object file is to be created under the selected file name.

Wildcard characters

Not allowed.

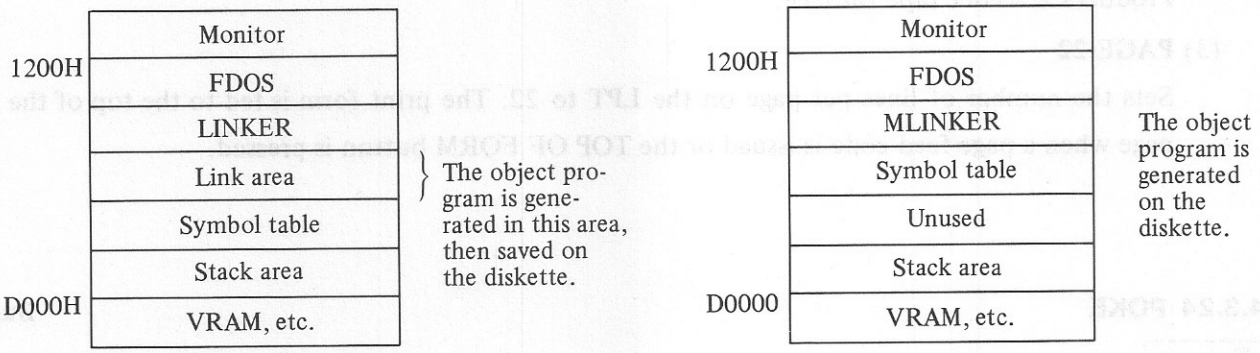
Examples

2 > MLINK STARTREK

Programming notes

(1) The MLINK command can be used in the same manner as the LINK command except that it cannot specify the table size (TBL$hh).

(2) The LINK command can generate an object file of up to approx. 36K bytes. The MLINK command is used when the file exceeds this size to generate object files of up to approx. 46K bytes. However, the MLINK command takes twice as long as the LINK command to generate an object file because the MLINK command links relocatable programs using a 2-pass system. The following diagrams show memory maps applicable to execution of the LINK and MLINK commands.



4.3.22 MON

Built-in

Format

MON

Function

The MON command returns control to the monitor.

Programming notes

Control is transferred to FDOS from the monitor with the following monitor command.

＊ J 1200

4.3.23 PAGE

Transient

Format

PAGE output-device or **PAGE n**

Function

The PAGE command carries out a paging operation on the output device specified by output-device, or sets the number of lines per page on LPT.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

(1) **PAGE** or **PAGE $ LPT**

Carries out a form feed on LPT.

(2) **PAGE $ PTP**

Produces a feeder tape on PTP.

(3) **PAGE 22**

Sets the number of lines per page on the LPT to 22. The print form is fed to the top of the next page when a page feed code is issued or the TOP OF FORM button is pressed.

4.3.24 POKE

Built-in

Format

POKE $nnnn, data1, ..., $uuuu, dataN

Function

Stores data1 consisting of an even number of digits in and from address $nnnn (4-digit hexadecimal number) on, ... and stores dataN consisting of an even number of digits in and from address $uuuu on. Any address is accessible. The maximum length from POKE to data N is 160 characters including 0DH, space, etc.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

POKE $CF0D, 2010, $CF0F, 40

Stores 20 in address $CF0D, 10 in $CF0E and 40 in $CF0F.

POKE $CF0D, 1235678, 12, $CF0F, 40

Not allowed

4.3.25 PROM

Transient

Format

PROM

Function

The PROM command converts the format of the object file to an appropriate PROM writer format.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Example

(1) **PROM**

Invokes the PROM formatter program and enters the command mode. Refer to the "PROM Formatter" manual for further information.

4.3.26 RENAME

Built-in

Format

RENAME oldname1, newname1, ..., oldnameN, newnameN

Function

The RENAME command renames specified files.

Default file mode

.ASC

Switches

None.

Wildcard characters

An asterisk may be used to specify the file mode (.＊).

Examples

(1) **RENAME TEST1, TEST2**

renames TEST1.ASC to TEST2.ASC.

(2) **RENAME $FD2 ; TEST1 . OBJ, TEST2, TEST3 . RB, TEST4**

Renames TEST1.OBJ on the diskette in FD2 to TEST2.OBJ and TEST3.RB on the diskette in the default drive to TEST4.RB.

Programming notes

(1) Files with the file attribute W or P cannot be renamed.

(2) The command RENAME $FD2;TEST1, $FD2;TEST2 cannot be executed since $FDn specified for the old name applies to the new name, which is illegal.

(3) The command RENAME TEST1.LIB, TEST2.RB cannot be executed since the file modes of the old and new names disagree.

(4) The command RENAME TEST.LIB, TEST2 can be executed normally. The new name is assigned the file mode of the old name.

4.3.27 RUN

Built-in

Format

RUN filename or file name

Function

The RUN command executes the program in the object file specified by the argument.

Default file mode

.OBJ, .SYS

Switches

None.

Wildcard characters

None.

Example

(1) **RUN TEST**

Executes the program TEST.OBJ. When its loading address is such that it overwrites the FDOS area, the system issues the message

destroy FDOS ?

on the CRT. When the programmer press the Y key, the system loads the program, overwriting the FDOS area and executing it. When the programmer presses the N key, the system issues the error message "memory protection" and waits for a new FDOS command.

(2) **1 > TEST**

Accesses the drive 1 to seek .SYS mode file and executes it if found. If not found, error occurs.

(3) **2] TEST**

Accesses drive 2 to seek program TEST.SYS and executes it if found. If not found, it seeks TEST .OBJ and executes it if found. If not found, error occurs.

Programming notes

The meanings of the prompt symbols (> and]) are shown below.

Command	filename	RUN filename	RUN $FDn filename	RUN $nnnn
File mode	. SYS . OBJ	. OBJ	. OBJ	
Prompt >	Accesses the drive 1 to seek . SYS mode file and executes it if found. If not found, error occurs.	Accesses the default drive to seek . OBJ mode file and executes it if found. If not found, error occurs.	Accesses $FDn to seek . OBJ mode file and executes it if found. If not found, error occurs.	Calls address $nnnn.
Prompt]	Accesses the default drive to seek . SYS mode file and executes it if found. If not found, it seeks . OBJ mode file and executes it if found. If not found, error occurs.	Same as above.	Same as above.	Same as above.

4.3.28 SIGN

Transient

Format

SIGN $FDn

Function

The SIGN command defines or changes the password and/or volume number of the diskette in the specified drive.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Example

(1) SIGN

Old sign ? **SHARP** ← Proceeds to the next step if the password entered matches the old password.
New sign ? **MZ-80**
New volume No ? **79**
The above interaction changes the password from "SHARP" to "MZ-80" and defines the volume number as 79.

4.3.29 STATUS

Transient

Format

STATUS devicename, $nnnn

Function

The STATUS command displays or sets the control status of the specified device. The control status information is used to initialize the I/O controllers. Refer to "User I/O Routine" in Appendix for detali.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

(1) STATUS $USR1

Displays the control status of USR1 on CRT.

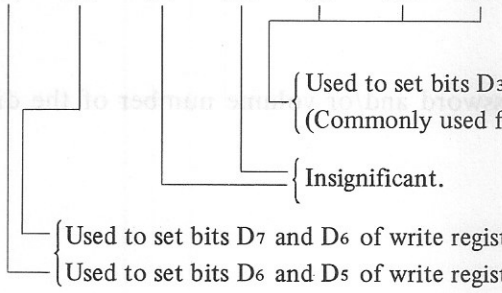
(2) STATUS $LPT, $0000 (initial value = $0000)

00 ..... normal mode
12 ..... double-size mode
14 ..... reduced mode

(3) STATUS $SIA, $xxyy (initial value = $0DCC)

xx : Line end mark

yy: D7 D6 D5 D4 D3 D2 D0



The write registers are shown below. For details, refer to the Z-80 SIO Technical Manual.

Write register 3

D7	D6	D5	D4	D3	D2	D1	D0	
|	|	1	0	0	0	0	X	
0	0	Rx	5 BITS/CHARACTER					Can be set by the STATUS command when SIA or SIB is used.
0	1	Rx	7 BITS/CHARACTER					
1	0	Rx	6 BITS/CHARACTER					
1	1	Rx	8 BITS/CHARACTER					

Write register 4

D7	D6	D5	D4	D3	D2	D1	D0	
0	1	0	0	|	|	|	└	PARITY ENABLE/DISABLE
						└		PARITY EVEN/ODD
				0	0			SYNC MODES ENABLE
				0	1	1		STOP BIT/CHARACTER
				1	0	1½		STOP BITS/CHARACTER
				1	1	2		STOP BITS/CHARACTER

Can be set by the STATUS command when SIA, SIB, SOA or SOB is used.

Write register 5

D7	D6	D5	D4	D3	D2	D1	D0	
X	|	|	0	X	0	X	0	
	0	0	Tx	5 BITS/CHARACTER				Can be set by the STATUS command when SOA or SOB is used.
	0	1	Tx	7 BITS/CHARACTER				
	1	0	Tx	6 BITS/CHARACTER				
	1	1	Tx	8 BITS/CHARACTER				

Programming note

This command is available for the serial I/O devices ($SIA, $SIB, $SOA and $SOB), $LPT and user devices ($USR1 to $USR4). Any STATUS command set for $PTR, $KB, $CRT, $FD1 to $FD4, $CMT, $MEM or $PTP will be invalid.

4.3.30 TIME

Built-in

Format

TIME mm : dd : ss

Function

The TIME command sets or displays the time of the system clock.

Default file mode

None.

Switches

Global switch /P: Specifies that the time is to be printed on LPT.

Wildcard characters

None.

Examples

(1) **TIME 20 : 30 : 40**

Sets the system clock to 20 hours, 30 minutes and 40 seconds.

(2) **TIME**

Displays the current time on CRT.

(3) **TIME/P**

Prints the current time on LPT

4.3.31 TYPE

Built-in

Format

TYPE filename1, ..., filenameN

Function

The TYPE command outputs contents of the files specified by the arguments on the CRT or LPT device.

Default file mode

. ASC

Switches

Global switch /P: Specifies that the file contents are to be printed on the LPT device.

Wildcard characters

Allowed.

Examples

(1) **TYPE TEST**

Displays the contents of source file TEST . ASC on CRT.

(2) **TYPE/P TEST1, TEST2**

Prints the contents of source files TEST1 . ASC and TEST2 . ASC on LPT.

4.3.32 VERIFY

Transient

Format

VERIFY sourcefile 1, destinationfile 1, ..., sourcefileN, destinationfileN

Function

The VERIFY command compares the contents of the source and destination files specified by the arguments and displays any mismatching contents on a line basis (if their file mode is .ASC) or on a byte basis (if the file mode is other than .ASC).

Default file mode

. ASC

Switches

Global switch /P: Specifies that the matching results are to be printed on LPT.

Wildcard characters

Allowed for source files (see example (4) below).

Examples

(1) **VERIFY TEST1, TEST2**

Matches source files TEST1.ASC and TEST2.ASC and displays mismatching lines on CRT.

(2) **VERIFY/P $CMT ; XYZ, $FD2 ; TEST**

Matches source file XYZ.ASC on CMT with source file TEST.ASC on the diskette in FD2 and prints the results on LPT.

(3) **VERIFY $CMT, $FD2**

Matches the first file on CMT with the file on the diskette in FD2 which has the same name as the file on CMT. An error is generated if file on CMT has no file name.

(4) **VERIFY $CMT ; TEST*, $FD2**

Matches the first file on CMT whose name matches TEST* with the file that name on the diskette in FD2. Note that only the first file whose file name matches TEST* is taken.

4.3.33 XFER

Built-in

Format

XFER sourcefile 1, destinationfile 1, ..., sourcefileN, destinationfileN

Function

The XFER command transfers the contents of source files to destination files.

Default file mode

. ASC

Switches

None.

Wildcard characters

Allowed for the source files (see example (5) below).

Examples

(1) **XFER TEST1, TEST2**

Transfers the contents of source file TEST1.ASC to TEST2.ASC.

(2) **XFER $PTR, $LPT**

Reads the file on PTR and prints it on LPT.

(3) **XFER $CMT ; XYZ.OBJ, $FD2**

Reads object file XYZ.OBJ from CMT and creates object file XYZ.OBJ on $FD2.

(4) **XFER $CMT, $FD2**

Reads in the first file on CMT and creates a file with that file name on the diskette in FD2. An error is generated if file on CMT has no file name.

(5) **XFER $CMT ; TEXT＊, $FD2**

Reads in the first file on CMT whose file name matches file name TEST＊ and creates a file with the same name on the diskette in FD2. Note that only the first source file on CMT whose file name matches TEST＊ is taken.

(6) **XFER $KB, TEST**

Reads a file from the system keyboard and creates source file TEST.ASC. The file read from the keyboard is terminated by pressing the BREAK key.

(7) **XFER $FD2 ; ＊. ASC, $FD3**

Transfers all source files on the diskette in FD2 to that in FD3. The source drive must not contain files with the file attribute R or P.

(8) **XFER ＊. ＊ , FD2**

Transfers all files on the diskette in the current default drive to that in FD2. The source drive must not contain files which have the file attribute R or P.

4.4 FDOS Command Summary

The FDOS commands are broadly divided into built-in commands (**Table 4-1**) and transient commands (**Table 4-2**). Transient commands are implemented in relocatable file form on the FDOS diskette. They are loaded into the transient area in main memory by the boot linker and linked to the FDOS main program as required.

In the command format in **Table 4**, the items enclosed in brackets are optional.

Table 4-1 Built-in commands

BOOT

Terminates the FDOS and activates system IPL.
Example: BOOT↵

CHATR sign, filenamel, attribute [, ...filenameN, attribute]

Matches the password's sign and changes the file attribute(s) of the matching file(s) identified by filename to attribute(s).
P: Permanent file R: Read inhibit
0: No protection W: Write inhibit

Examples: CHATR KEY, ABC, 0, XYZ, P↵ : Deletes the file attribute of file ABC and changes the file atrribute of file XYZ to PERMANENT if matches occur with the password KEY.

CHATR KEY, $FD2 ; UVW, R↵ : Changes the file attribute of file UVW in FD2 to READ INHIBIT if a match occurs with the password KEY.

CHATR↵ : This allows the programmer to interactively specify the password, file name and attribute.

DATE [MM . DD . YY]

Displays the current date or sets the specified date in month, date, year format. The set information is used as file information when new files are created.
Global switch /P : Specifies that the date is to be printed on the LPT.
Examples: DATE/P↵ : Lists the current date on the LPT.
DATE 12 . 25 . 81↵ : Sets the current date to December 25, 1981.

DELETE filenamel [, ..., filenameN] (?, ✳)

Deletes the file(s) specified by filename(s).
Global switch /C : Specifies that each file name is to be displayed on the screen for verification. The programmer must enter Y to delete it or N to suppress deletion.

Examples: DELETE ABC . ✳↵ : Deletes all files identified by ABC . ✳.
DELETE/C A ✳ . ✳↵ : Displays files identified by A ✳ . ✳ on the screen for verification before deletion.
filename : ABC.ASC deleted ← Indicates that the file is deleted since "Y" is entered.
filename : ABC.RB ← Indicates that the file is not deleted "N" is entered.
filename : AXY.OBJ permanent ← Indicates that the file is not deleted because it is assigned the PERMANENT file attribute.

DIR [$FDn] or [filename] (?, ✳)

Displays file information in the directory specified by $FDn or of the file specified by filename on the screen.
Global switch/P : Specifies that the file information is to be output to LPT. The file information is displayed on the screen when this switch is not specified.

Examples: DIR↵ : Displays all file information in the current directory on the screen.
DIR/P $FD2↵ : Outputs all FD2 file names to LPT and switches the currently logged disk to FD2.
DIR $FD2 ; ABC. ✳↵ : Displays the file information of files in FD2 identified by ABC. ✳.

Table 4-1 Built-in commands cont.

EXEC filename

Executes the contents of the file identified by filename as FDOS commands.
Example: EXEC ABC . ASC ↵ : Sequentially executes the FDOS commands in file ABC.

FREE [$FDn]

Lists statistical information about the disk identified by$FDn on the screen or on the LPT.
Example: FREE $FD2 ↵
$FD2 master left : XXXX used : YYYY
Indicates that the diskette on FD2 is a master diskette, that the number of unused sectors is XXXX and that the number of used sectors is YYYY.

HCOPY control-code

Copies a data frame from the CRT screen to the LPT.
Example: HCOPY ↵

MON

Terminates FDOS processing and returns control to the monitor.
Example: MON ↵

POKE $nnnn, date [, ..., $uuuu, dataN]

Stores data in the specified addresses in memory.
Example: POKE $000D, 2010, $000F, 40 ↵

RENAME oldname1, newname1 [, ..., oldnameN, newnameN]

Renames the file specified by oldname to newname.
Examples: RENAME ABC, XYZ ↵ : Renames file ABC to XYZ.
RENAME ABC, DEF, UVW, XYZ ↵: Renames file ABC to DEF and UVW to XYZ.

RUN filename

Executes the program in the object file identified by filename.
Example: RUN ABC ↵ : Executes the program in file ABC, assuming it to be ABC.OBJ.

TIME [HH : M : SS]

Displays the current time or sets specified time in hour, minute, second format.
The current time is set to 00 : 00 : 00 upon system start.
Global switch/P : Specifies that the current time is to be listed on the LPT.
Examples: TIME/P ↵ : Lists the current time on the LPT.
TIME 16 : 30 : 30 ↵ : Sets the current time to 16 : 30 : 30

TYPE filename1 [, ..., filenameN] (?, ＊)

Lists the contents of the file(s) identified by filename(s) on the screen or on LPT.
Global switch/P : Lists the file contents on LPT.
Examples: TYPE ABC, DEF ↵ : Displays the contents of files ABC and DEF on the screen.
TYPE/P $FD3 ; XYZ ↵ : Lists the contents of file XYZ in FD3 on LPT.
TYPE $PTR ↵ : Reads paper tape data from PTR and displays it on the screen.

XFER sourcefile1, destinationfile2 [, ..., sourcefileN, destinationfileN] (sourcefile only ?, ＊)

Transfers the source file(s) to the destination file(s).
Examples: XFER ABC, XYZ ↵ : Copies file ABC to XYZ.
XFER $PTR, DEF ↵ : Transfers the file at the PTR to file DEF.
XFFR XYZ, $PTP/PE ↵ : Transfers file XYZ to the PTP with even partiy in ASCII code.

Table 4-2 Transient commands

ASM filename

Assembles the source file identified by filename and produces a relocatable file and an assembly listing.
Global switch (none) : Specifies that the relocatable file is to be output.
Global switch/N : Suppresses generation of the relocatable file.
Local switch/O : Specifies that the relocatable file is to be output with the specified file name.
Local switch/E : Specifies that error statements are to be output to the specified file.
Local switch/L : Specifies that the listing is to be directed to the specified file.
Examples: ASM ABC ↵ : Assembles source file ABC and generates relocatable file ABC.RB.
ASM/N ABC, $CRT/E ↵ : Assembles source file ABC and displays error statements on the screen (no relocatable file is created).
ASM ABC, XYZ/O, $LPT/L ↵ : Assembles source file ABC and generates relocatable file XYZ.RB and an assembly listing on the LPT.
ASM ABC, $FD2 ; XYZ/L, $LPT/E ↵ : Assembles source file ABC outputs the assembly listing to file XYZ.ASC in FD2 and outputs error statements on the LPT.

ASSIGN devicename, address

Sets the address of a user device drive routine.
Example: ASSIGN $USR1, $B000 ↵ : Sets the drive routine address of user device $USR1 to B000 (hexadecimal).

BASIC filename

Invokes the BASIC compiler to compile the source program identified by filename.
Example: BASIC XYZ ↵ : Invokes the BASIC compiler, compiles source file XYZ.ASC and generates relocatable file XYZ.RB.

CONVERT

Converts a file generated with the SA-5000 series BASIC interpreter or the D-BASIC SA-6000 series into a file which can be used under FDOS, or converts a file generated with FDOS into a file which can be used under the SA-5000 series BASIC interpreter or the D-BASIC SA-6000 series.
Example: CONVERT ↵

COPY

Copies the files on the diskette in drive 1 to the diskette in drive 2. The system matches the passwords in these diskettes before carrying out a copy operation.
Example: COPY ↵

DEBUG filename [, ..., filenameN]

Invokes the symbolic debugger and links and loads relocatable file(s).
Global switch /T : Specifies that the symbol table information is to be output.
Global switch /P : Specifies that the listing is to be directed to the LPT (the listing is displayed on the screen if omitted).
Local switch /O : Specifies that the object file is to be generated with the specified file name.
Example: DEBUG ABC, DEF ↵ : Invokes the symbolic debugger, links and loads relocatable files ABC and DEF and waits for a symbolic debugger command.

EDIT [filename]

Loads the text editor and reads in the file (if specified). The file must be an ASC mode file.
Examples: EDIT ↵ : Loads the text editor and waits for an editor command.
EDIT $FD2 ; ABC ↵ : Loads the text editor and reads in file ABC from FD2.

Table 4-2 Transient commands cont.

FORMAT [$FDn]

Initializes the diskette in $FDn in the system format. The password set by the SIGN command is checked before execution.

Examples: FORMAT ↵ : Initializes the currently logged-on diskette.
FORMAT $FD2 ↵ : Initializes the diskette in FD2.

LIBRARY filename1 [, ..., filenameN]

Links specified file(s) into a library file.

Global switch (none) : Specifies that the link information is to be displayed on the screen.
Global switch /P : Specifies that the link information is to be printed on the LPT.

Examples: LIBRARY ABC, DEF, ↵ : Links relocatable files ABC and DEF and stores their contents into library file ABC.LIB
LIBRARY ABC, DEF, XYZ/O ↵ : Links relocatable files ABC and DEF and stores their contents into library file XYZ.LIB.

LIMIT address

Sets or changes the end address of the memory area managed by FDOS.

Examples: LIMIT $B000 ↵ : Sets the FDOS area to B000 (hexadecimal).
LIMIT MAX ↵ : Sets the FDOS area to the maximum available address.

LINK filename1 [, ..., filenameN]

Links relocatable files identified by filename1 through filenameN and outputs an object file with a link table listing.

Global switch /T : Specifies that the symbol table information is to be listed.
Global switch /P : Specifies that the listing is to be directed to the LPT (the listing is displayed on the screen if the switch is omitted).
Global switch /S : Specifies that a system file is to be generated.

Examples: LINK ABC, DEF ↵ : Links relocatable files ABC and DEF and outputs object file ABC.OBJ
LINK/T/P ABC, DEF, XYZ/O↵: Links relocatable files ABC and DEF and outputs object file XYZ.OBJ with the link table information on the LPT.

LOAD filename

Loads the object file identified by filename into the area immediately following the area established by the LIMIT command.

Example: LOAD ABC.OBJ ↵ : Loads object file ABC.OBJ into memory.

MLINK filename1 [, ..., filenameN]

Links relocatable files identified by filename1 through filenameN and outputs an object file with a link table listing. This command can link files to generate an object file of up to 30K bytes, although the LINK command can only deal with up to 20K bytes.

Global switch /T : Specifies that the symbol table information is to be listed.
Global switch /P : Specifies that the listing is to be output on the LPT (the listing is displayed on the screen if this switch is omitted).

Example: MLINK ABC, DEF ↵ : Links relocatable files ABC and DEF and outputs object file ABC.OBJ.

Table 4-2 Transient commands cont.

PAGE [output-device] or nn

Performs a form feed operation on the output device identified by output-device, or sets the number of lines per page on the LPT.

Examples: PAGE ↵ : Moves the print position to the home position of the printer form.

PAGE 22 ↵ : Sets the number of lines per page on the LPT to 22. The print form is fed to the top of the next page when a page feed code is issued or the TOP OF FORM button is pressed.

PROM

Generates formatted code on the paper tape punch from an object file. Applicable PROM writers are those which are supplied by Britronics, Intel, Takeda and Minato Electronics.

Example: PROM ↵

SIGN [$FDn]

Changes the password of the diskette in $FDn.

During a diskette copy or formatting operation, the system checks the programmer-specified password with that stored in the diskette directory for a match and carries out the specified operation only when a match occurs.

Example: SIGN ↵ : Changes the password of the diskette currently logged on.

STATUS devicename, status

Sets the status of the I/O device identified by devicename to status.

Example: STATUS $SIA, $1234 ↵ : Sets the control status of serial input port A to 1234 (hexadecimal).

VERIFY filename1, filename2 [, ..., filenameN-1, filenameN] **(?, ✳ only for filename1, ..., filenameN-1])**

Compares the contents of files filename1 through filenameN.

Global switch /P : Specifies that the results of the comparison are to be listed on the LPT.

Example: VERIFY $CMT, $FD2 ; ABC ↵ : Compares the first file on the cassette tape with source file ABC in FD2.

4.5 System Error Messages

There are four system error message formats.

— ERR: error message

Pertains mainly to coding errors. Issued when invalid commands are detected.

— ERR filename (device name) : error message

Indicates errors pertaining to file or device specifications.

— ERR logical number: error message

Indicates errors pertaining to logical number specifications.

— ERR logical number file name (device name): error message

Indicates errors pertaining to logical number specifications and file (or device) specifications.

The system error messages are listed below. The error numbers are not output.

ERR-	1	syntax	
	2	il command	
	3	il argument	
	4	il global switch	
	5	il data	
	6	il attribute	; Illegal file attribute found
	7	different file mode	
	8	il local switch	
	9	il device switch	
	10		
	11	no usable device	; Device unavailable
	12	double device	
	13	directory in use	
	14		
	15		
	16	not enough arguments	
	17	too many argument	
	18		
	19		
	20	no memory space	
	21	memory protection	
	22	END ?	
	37	Break	
	38	system id	; Diskette not conforming to FDOS format.
	39	System error	; System malfunction, user program error, diskette replaced improperly, etc.

50	not found	
51	too long file	; File size exceeds 65535 bytes
52	already exist	
53	already opened	; The file has been already opened or
54	not opended	the logical number is already used.
55	read protected	
56	write protected	
57	permanent	
58	end of file	
59	no byte file	
60	not ready	
61	too many files	; Number of files exceeds 96
62	disk volume	; Diskette replaced improperly
63	no file space	
64	unformat	; Diskette unformatted
65	FD hard error	; Hardware related disk error
66	il data	
67	no usable diskette	
68	(sub)master diskette	
69	mismatch sign	
70	il file name	; Invalid file name
71	il file attribute	; Invalid file attribute
72	il file type	; Invalid file type
73	il file mode	; Invalid file mode
74	il lu#	; Invalid logical number
75	not ready	; Printer error
76	alarm	
77	paper empty	
78	time out	; Paper tape reader or punch error
79	parity	
80	check sum	
89	lu table overflow	; Attempt made to open too many files
90	source ?	
91	destination ?	
92	can't xopen	
93	too long line	; Line exceeding 128 bytes
94	end of record	
95	diff record length	

5. DISKETTE HANDLING

5.1 Verification during Write

The data written on a diskette is verified automatically. This automatic verification function can be overridden to increase execution speed of the COPY command, XFER command and the like, but reliability is reduced.

Automatic verification during a write can be stopped by executing POKE $1040,FF and restarted by executing POKE $1040,00.

5.2 Diskette Replacement

- Diskettes can be replaced at anytime when the FDOS is in the command wait state.
- Never replace a diskette when the drive motor is rotating or the contents of the diskette may be destoryed.
- If a diskette is replaced during a period when it should not be replaced, the error message "disk volume" appears on the screen but the contents of the diskette are not destroyed; the volume number is checked at this time.

  Therefore, all diskettes used should be assigned different volume numbers. (The volume number can be displayed with the FREE command and can be changed with the SIGN command).

5.3 Destruction of the Contents of Diskettes

The contents of a diskette may be destroyed under the following circumstances.

a. If a diskette is replaced while the drive motor is rotating.
b. If the user program runs without control and accesses the diskette.
c. If message "System error, re-boot FDOS" appears.
d. If message "FD hard error" appears.
e. If message "unformat" appears when a formated diskette is used.

Take the following measures if any of the above occur.

- Reload the FDOS. (Press the ipl switch with the master diskette loaded).
- Transfer undamaged files from the destroyed diskette to a normal one with the XFER command. Try each file several times even if it cannot be transferred on the first try.
- Reformat the destroyed diskette.

It is recommended that copies be made of data diskettes using the COPY command to prevent their contents from being destroyed for any reason (e.g., diskette damage, careless execution of a DELETE command, etc.).

6. MUTUAL CONVERSION

Mutual conversion between files generated by different system programs are possible for the following combinations of files using the conversion procedure shown:

Possible Combinations of Files

File 1					File 2				Procedure
Model	System Program	Media	Mode		Model	System Program	Media	Mode	
A	BASIC	FD / CMT	BTX	↔	A	FDOS	FD / CMT	ASC	use CONVERT command
A	BASIC	FD / CMT	BSD	↔	A	FDOS	FD / CMT	ASC	use CONVERT command
A	BASIC	FD	OBJ	↔	A	FDOS	FD / CMT	OBJ	use CONVERT command
A / K	SP-2xxx	CMT	ASC / OBJ	↔	A	FDOS	FD	ASC / OBJ	compatible (use XFER command)
K	FDOS	CMT	ASC / OBJ	↔	A	FDOS	FD	ASC / SBJ	compatible (use XFRR command)
B	FDOS	FD	All	↔	A	FDOS	FD	All	compatible (use XFER command)
B	BASIC	FD	BTX	↔	A	FODS	FD / CMT	ASC	use CONVERT command
B	BASIC	FD	BSD	↔	A	FDOS	FD / CMT	ASC	use CONVERT command
B	BASIC	FD	OBJ	↔	A	FDOS	FD / CMT	OBJ	use CONVERT command
K	BASIC	CMT	BTX	→	A	BASIC	CMT	BTX	use convert-tape (MZ-8AT01)
B	BASIC	FD	All	↔	A	BASIC	FD	All	compatible

A : MZ-80A

K : MZ-80K

B : MZ-80B

BASIC : BASIC interpreter Sx-5xxx, Sx-6xxx.

FDOS : FDOS or BASIC compiler Sx-7xxx.

FD : Floppy disk.

CMT : Cassette tape.

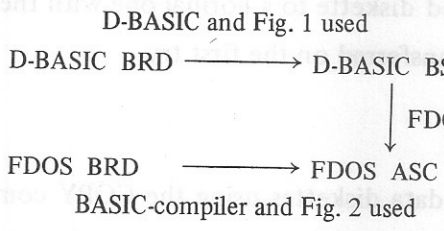
BTX : BASIC interpreter text file.

BSD : BASIC interpreter sequential data file.

ASC : ASCII file.

OBJ : Object file.

when converting BRD generated by D-BASIC to File of a form acceptable by FDOS:




```
10 REM  BRD → BSD sample conversion program.
20 INPUT "RND FILE ? ";R$
30 INPUT "SEQ FILE ? ";S$
40 XOPEN #1,R$: WOPEN #2,S$
50 I=1
60 INPUT #1(I),A$: IF EOF(#1) THEN CLOSE : END
70 PRINT #2,A$: I=I+1: GOTO 60
```

Fig. 1

```
10 REM  BSD → BRD sample conversion program.
20 INPUT "SEQ FILE ? ";S$
30 INPUT "RND FILE ? ";R$
40 ROPEN #1,S$: XOPEN #2,R$
50 I=1: D$=CHR$($0D)
60 A$=""
70 INPUT #1,B$: IF EOF(#1) THEN CLOSE : END
80 A$=A$+B$: L=LEN(A$)
90 IF L>32 THEN PRINT "ERROR": CLOSE : END
100 IF L<32 THEN A$=A$+D$: L=L+1
110 IF L<32 THEN 70
120 PRINT #2(I),A$: I=I+1: GOTO 60
```

Fig. 2

The following cassette based system programs have thus far been released.

- **MACHINE LANGUAGE SP-2001**
- **RELOCATABLE LOADER SP-2301**
- **SYMBOLIC DEBUGGER SP-2401**
- **EDITOR-ASSEMBLER SP-2202, SP-2102**

These system programs generate source files (with file mode.ASC), relocatable files (with file mode RB), object files (with file mode .OBJ) and debug mode save files (i.e., object files with symbol tables). **Of these, source files and object files can be transferred to FDOS diskettes.**

The procedure for transferring a cassette file to an FDOS file is as follows.

When the file name consists of characters which are usable with FDOS:

XFER $CMT, $FDn (n = 1 – 4)

When the file name includes characters which are not allowed by FDOS, a new file name must be assigned as follows:

XFER $CMT, $FDn; filename (n = 1 – 4)

When an assembly source file is to be transferred, use the following procedures to determine whether or not pseudo instruction REL is used: load the file with the FDOS text editor and search for REL with the S command. Delete all REL instructions; this is because FDOS system programs do not require REL. Next, assemble the file from which REL instructions have been deleted to generate a relocatable file with the FDOS assembler. The object file is obtained by relocating it.

Object files generated by cassette based system programs can be transferred to an FDOS file and they can be executed by the following command.

RUN $FDn; filename

The following message is displayed on the CRT screen when the specified object file has a loading address which results in destruction of the FDOS area.

DESTROY FDOS?

Pressing the [Y] key at this time performs the transfer operation, destroying the FDOS area; pressing the [N] key stops the operation and returns the system to the FDOS command wait state.