PREFACE

Thank you for buying the Sharp Personal Computer MZ-80 series FDOS.

To make the best use of the FDOS, read the instruction manual thoroughly and perform the operations described correctly; this will enable you to make most effective use of the system.

— Reserving 48K bytes of RAM for FDOS will allow a larger effective memory configuration for operation of the text editor and the relocating loader.

— The master diskette cannot be replaced after it is purchased; therefore, be sure to use the COPY command to create a submaster diskette for normal use.

— It is particularly important to read and understand the explanations of the following commands before using FDOS.

- FORMAT command (page 49 of the System Command manual).
  Before using FDOS with a new diskette, it must be formatted and initialized for FDOS. The file contents of diskettes initialized for use with other systems (e.g., SP-6015 or SP-6115) will be destroyed if used with this system. Likewise, diskettes initialized with FDOS cannot be used with other systems.
- COPY command (page 43 of the System Command manual).
  This command allows creation of submaster diskettes from the master diskette and of backup diskettes for slave diskettes.

— Since the FDOS operating instructions are divided into several parts, a guide is included to enable easy reference as needed. Full understanding of FDOS is not a prerequisite to making active use of it; refer to the guide as needed and your knowledge of the system will grow as you use it.

PRODUCT GUIDE

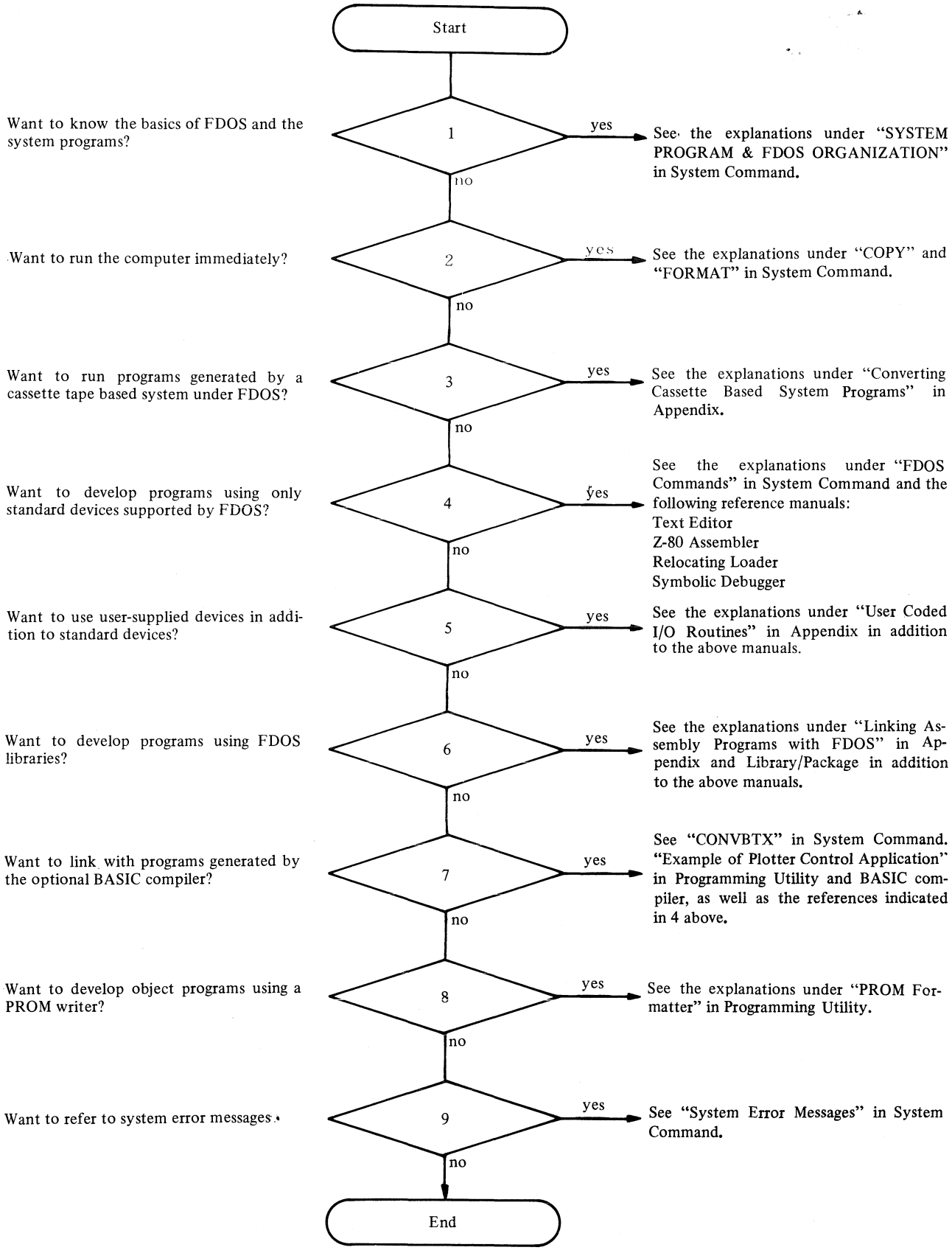
The following materials are included in this group of products.

"System Command" Instruction Manual
"Text Editor" Instruction Manual
"Z-80 Assembler" Instruction Manual
"Symbolic Debugger" Instruction Manual
"Relocating Loader" Instruction Manual
"Programming Utility" Instruction Manual
"PROM Formatter" Instruction Manual
"Example of Plotter Control Application"
"Library/Package" Instruction Manual
"Appendix"
"FDOS Master Diskette"

Also, the following files are included in FDOS Master Diskette. Refer to the various instruction manuals for details.

File name	Applicable command or manual	Function
ASM . SYS	ASM	Z-80 Assembler
ASSIGN . SYS	ASSIGN	Device definition
BASIC (OPTION) . SYS	BASIC	BASIC compiler (sold separately)
COPY . SYS	COPY	Copying diskettes
DEB . SYS	DEBUG	Symbolic debugging
DEBUG . SYS	DEBUG	Symbolic debugging
EDIT . SYS	EDIT	Text editing
FDOS . ASC	"Library/Package"	FDOS library source file
FDOS . LIB	"Library/Package"	Library file for the above
FORMAT . SYS	FORMAT	Formatting diskettes
LIBRARY . SYS	LIBRARY	Creating library files
LIMIT . SYS	LIMIT	FDOS management area declaration
LINK . SYS	LINK	Relocating loader
LOAD . SYS	LOAD	Loading object files
MON . ASC	"Library/Package"	MONITOR library source file
MON . LIB	"Library/Package"	Library file for the above
PLOTTER . ASC	"Programming Utility"	X-Y plotter control subroutines
PROM . SYS	PROM	PROM formatter
RELO . LIB	"Library/Package"	BASIC compiler library file
SIGN . SYS	SIGN	Password registration for diskettes
STATUS . SYS	STATUS	Device status control
VERIFY . SYS	VERIFY	Comparison of files
X-YDEMO . ASC	"Programming Utility"	X-Y plotter BASIC demonstration program

——Guide to Use of These Publications——



——Optional FDOS Program Products——

1. **BASIC Compiler SP-7715 (Previously released)**

   Requirements: FDOS and 48K bytes of RAM

   Major features: *Fast execution

   *FDOS commands can be invoked from BASIC programs.

   *Can be linked to assembly language programs.

   Compilation: Compiles a source file (source program) and generates a relocatable file (RB file) which can be linked and loaded with the FDOS LINK command.

   Compatibility: Programs developed by SP-5000 series or SP-6015 must be converted to the FDOS format by the FDOS CONVBTX command before compilation. Some BASIC commands (file handling commands) may differ in syntax. Excessively large programs may not be compilable (source programs are limited about to 10K bytes).

   Packaging: The BASIC compiler is available on cassette tape with a reference manual. The compiler should be copied onto the submaster diskette so that it can be run under FDOS control.

2. **Serial I/O Ports (to be released in the nearfuture)**

   Requirements: FDOS and 48K bytes of RAM

   Major features: *Two 8251 serial interface LSI devices.

   *Baud rate is switch-selectable.

   *Two RS232C I/O channels, one of which may be used for a current loop circuit.

   Packaging: *One interface board, its control programs (on cassette tape) and a reference manual.

   *The control programs should be copied onto the submaster diskette so that they can be run under FDOS control.

System Command

SHARP

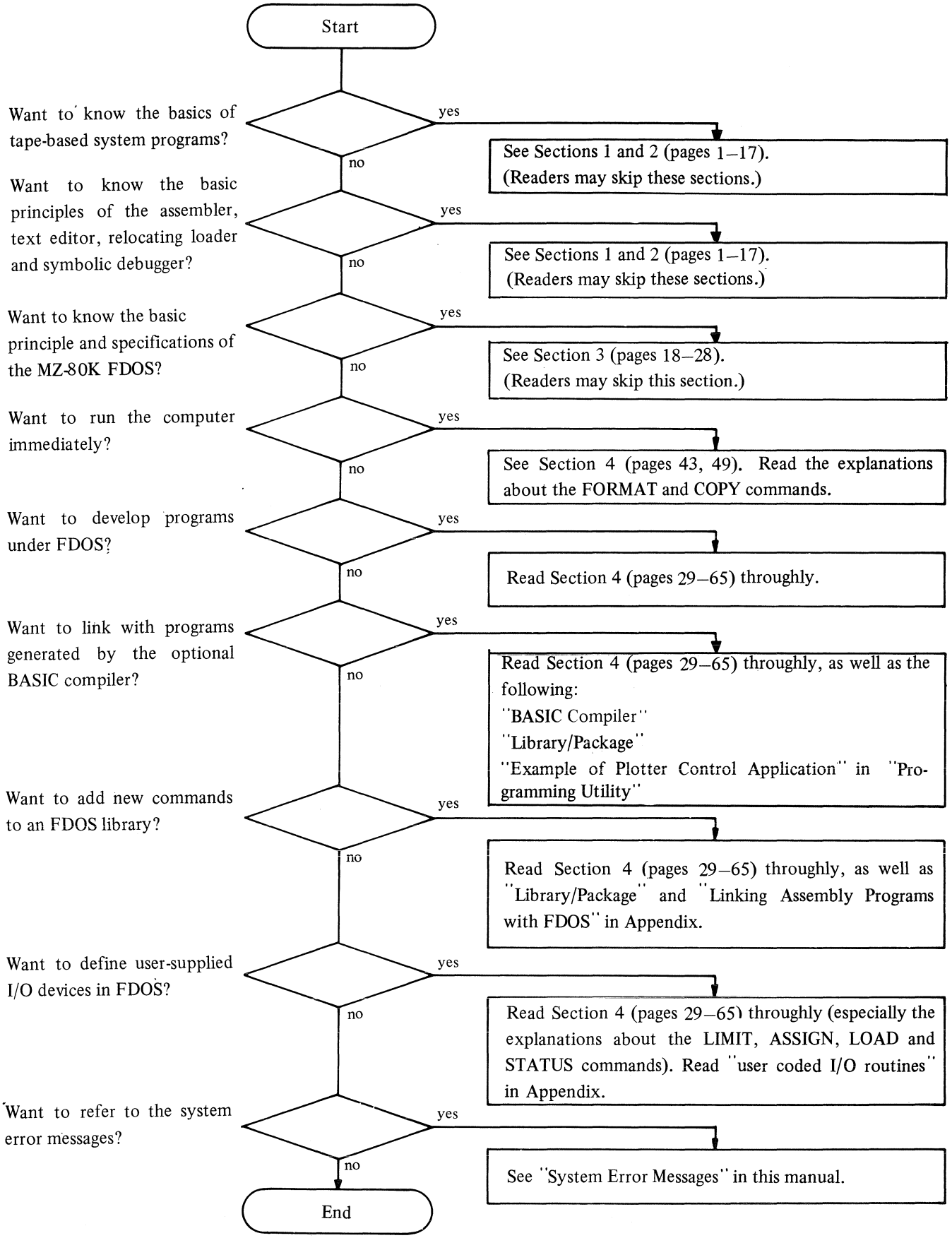
NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

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 provide 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 PROGAM ORGANIZATION (TAPE BASE)

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

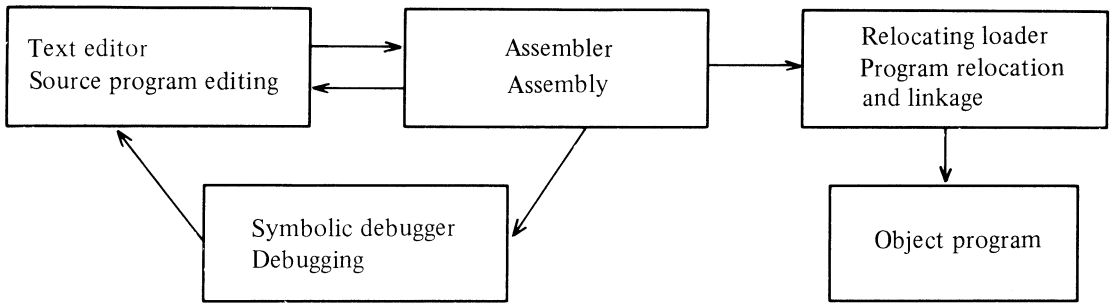


Fig. 1 Assembly phases

Figure 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 state. 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 assembler errors are corrected, the programmer inputs the relocatable binary file(RB file) output by the assembler to the symbolic debugger.

The symbolic debugger reads the relocatable binary file 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, suspend and resume program execution, and to display and alter registers 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 relocating loader.

Each system program always generates an output file for use in other system programs. Figure 2 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-80K can accomodate large-scale, serious application programs, thus enhancing its program development capabilities.

SHARP also supplies sytem backup software called the "editor assembler" which simplifies the very first stage of program development for relatively small source programs. A "PROM formatter" is also provided which punches object programs into paper tape in several formats for use with various PROM writers now on the market.

Table 1 lists the system program commands.

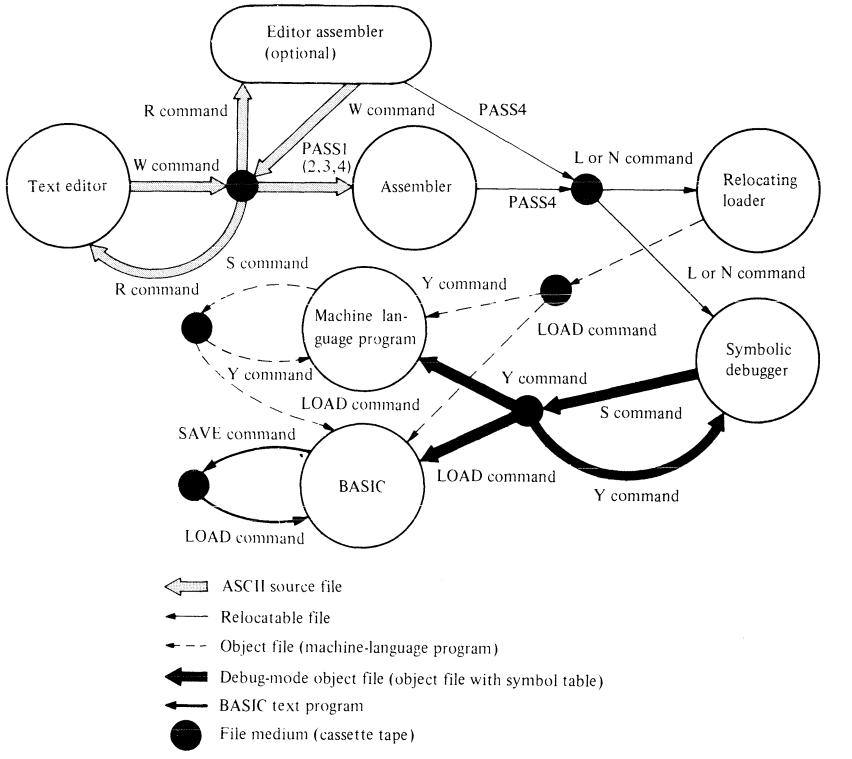


Fig. 2 Relationship of tape based system programs

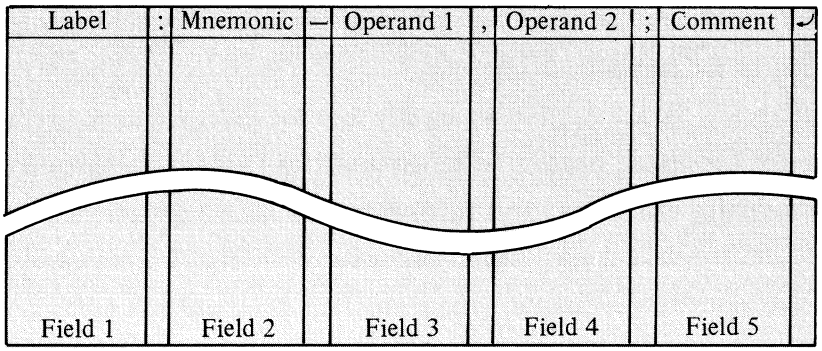


Fig. 3 Assembler coding format

2.1 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 that the operating principles and architecture of most computers tend to be alike. It is therefore possible to develop a general-purpose assembler for such microprocessors. 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 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 4) and a 2-dimensional operation table (Table 2) are introduced.

The symbol # in the instruction list represents a register and the symbol $ represents a numeric value or a label. 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 2, 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 requries 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 assembly operating procedure is summarized in Figure 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. The MZ-80K assembler is currently being used not only as a Z-80 self-assembler but also as cross assemblers for the Intel 8080A, Fujitsu MB8840 series (4-bit microprocessors), and NEC μCOM40 series (4-bit microprocessors).

Table 1 System program commands

(a) Assembler

Command	Function
PASS 1	Reads a source file and creates a symbol table. Since assembly is carried out in two modes, the programmer must specify one of the following modes during pass 1: CASSETTE: The source program is read into memory during each pass. RAM: The source program is read into the RAM area during pass 1; it is not read in the other passes.
PASS 2	Displays the assembly listing on the CRT screen. Display of the listing may be suspended and resumed with the SPACE key.
PASS 3	Outputs the assembly listing to the printer.
PASS 4	Outputs the relocatable binary code to the output file with a file name. In the CASSETTE mode, the assembler reads the source file to generate relocatable binary code in the RAM area before outputting it to a relocatable file.
!	Returns control to the monitor. The assembler is reentered by GOTO$2200 (cold start) or GOTO$222B (warm start).

(b) Text editor

Command		Function
Input commands	R (Read file) A (Append file)	Clears the edit buffer and inputs the input file specified by the file name. Appends the input file specified by the file name to the program in RAM at the position indicated by CP.
Type commands	T (Type) nT	Types the entire contents of the edit buffer. Types n lines from the CP position.
CP move commands	B (Begin) nJ (Jump) nL (Line) L nM (Move) Z	Moves the CP to the beginning of the edit buffer. Moves the CP to the beginning of line number n. Moves the CP to the beginning of the line n lines away from the line containing the CP. n may be a negative number. Same as the nL command with n = 0; that is, moves the CP to the beginning of the current line. Moves the CP n characters forward (n > 0) or backward (n < 0). Moves the CP to the end of the edit buffer.
Modify commands	C (Change) Q (Queue) I (Insert) nK (Kill) nD (Delete)	Searches for string 1 starting at the CP and, if found, substitutes it for string 2. Repeats the C command starting at the CP position and continuing until the end of the edit buffer is reached. Inserts a string at the CP position. The string is delimited by → marks. Deletes n lines from the edit buffer starting at the line containing the CP. Deletes n characters from the edit buffer starting at the CP position.
Search command	S (Search)	Searches for a string. After the string is found, the CP is relocated to the end of it.
Output command	W (Write)	Outputs the contents of the edit buffer to the output file specified by the file name.
Compare command	V (Verify)	Compares the contents of the file specified by the file name with the contents of the edit buffer.
Special commands	= . & (Clear) # ! (Goto Monitor)	Displays the total number of characters (including spaces and carriage returns) in the edit buffer. Displays the line number of the line containing the CP. Deletes the contents of the edit buffer. Switches the printer listing mode. Transfers control to the monitor. The editor is reentered by GOTO$1200 (cold start) or GOTO$1260 (warm start).

(c) Relocating loader

Command		Function
Link and load commands	L (Linking Load)	Given the assembly bias and loading address, loads a program unit from a relocatable file and generates absolute binary code in the link area.
	N (Next file)	Appends (links) the program unit in the next relocatable file to the absolute binary code in the link area.
	H (Height)	Displays the current assembly bias and loading address.
	T (Table dump)	Displays the contents of the symbol table.
Output command	S (Save)	Outputs the absolute binary code in the link area and the execution and data addresses to a named file.
Compare command	V (Verify)	Compares the contents of the saved file and the contents of the link area.
Special Commands	✳(Clear)	Clears the contents of the symbol table and sets the assembly bias and loading address to 0000.
	#	Switches the printer list mode.
	! (Goto Monitor)	Transfers control to the monitor. The relocating loader is reentered by GOTO$1200 (cold start) or GOTO$1260 (warm start).

(d) Symbolic debugger

Command		Function
Debugging commands	B (Break point)	Sets a breakpoint and a break count. The programmer can set a maximum of 9 breakpoints and a maximum count value of E in hexadecimal.
	& (Clear B.P.)	Clears all breakpoints.
	M (Memory dump)	Displays the contents of the specified memory block in the link area in hexadecimal representation. The command also permits memory alteration.
	D (memory list Dump)	Displays the contents of the specified block in the link area in hexadecimal representation with one instruciton on a line.
	W (data Write)	Writes data into the link area starting at the specified address.
	G (Goto)	Executes the program at the specified address.
	I (Indicative start)	Executes the program with the CPU registers loaded with the register buffer data, starting at the address designated by PC.
	A (Accumulator)	Displays the contents of register pairs AF, BC, DE and HL. The command allows register data alteration.
	C (Complementary)	Displays the contents of register pairs AF', BC', DE' and HL'. The command allows register data alteration.
	P (Program counter)	Displays the contents of special registers PC, SP, IX, IY and I. The command allows register data alteration.
	R (Register)	Displays the contents of all registers.
	X (data TRANSfer)	Transfers the contents of the specified memory block to the specified memory area.

File I/O commands	S (Save)	Saves the absolute binary code in the link area in the output file specified by the file name.
	V (Verify)	Compares the contents of the file specified by the file name with the contents of the link area.
	Y (Yank)	Loads the program unit in the debug mode form from the specified object file.

In addition to the above commands, the symbolic debugger includes the same link and load commands and special commands as the relocating loader.

```
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$)'     ; LIKE LD A, (IX + 15)
11  000D  2C284958
12  0011  2429
13  0013  F3                    DFFB    F3H       3 indicates the length of the instruction in bytes
14  0014  DD46                  DFFW    46DDH   } DD4600 is the main portion of the
15  0016  00                    DFFB    00H     } mnemonic code.
16  0017  03                    DFFB    03H       Atom type
17  0018  4C442023              DFFM    'LD #, (IY$)'     ; 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. 4 Instruction list (part)

Table 2 Two-dimensional operation table

Atom type	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					

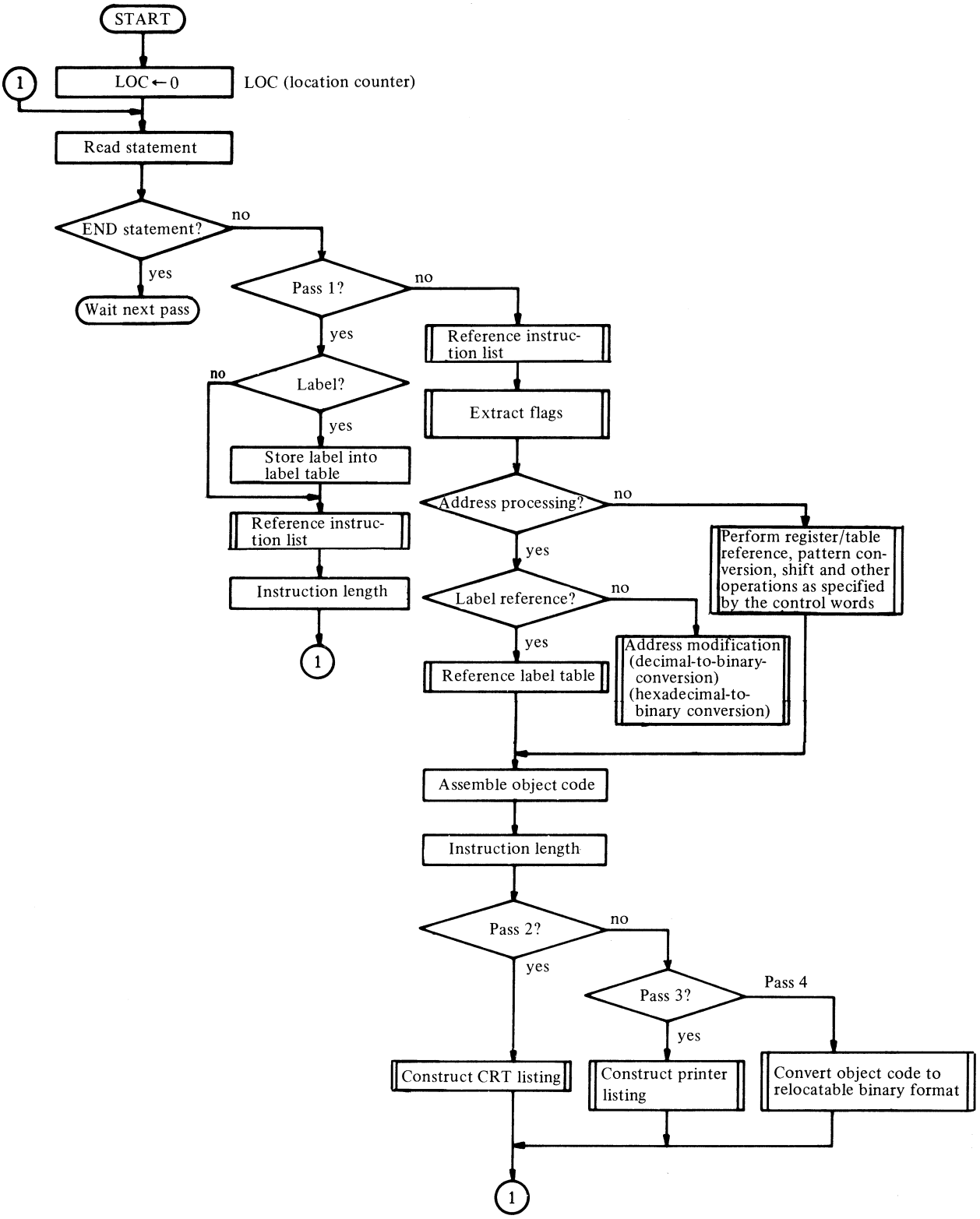


Fig. 5 General assembly flow (excluding assembler directive processing)

2.2 Text Editor Functions

The major functions of a text editor are to insert, delete and modify characters, word 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 users.

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 6. Therefore, the location to/from which a line is to be inserted/deleted can be 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 uses this interline pointer concept (see Table 1 (b)).

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 MZ-80K system also provides an editor-assembler (cassette based) which combines the features of the text editor and the assembler. Using the X command, the programmer can transfer to either program; this considerably reduces overhead time during the debugging session.

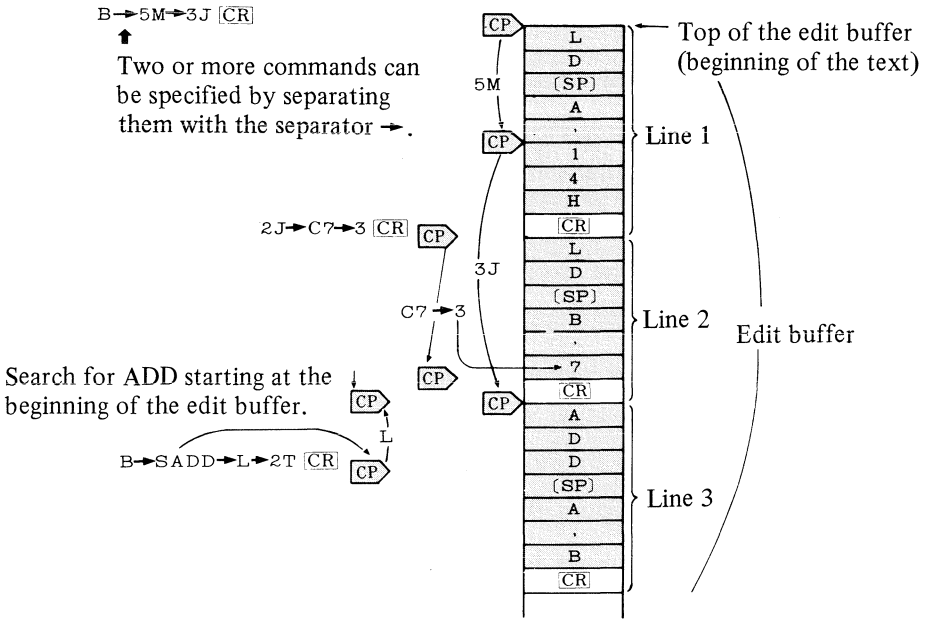


Fig. 6 Character pointer movement

2.3 Relocating Loader

The relocating loader loads and links two or more program units using external symbol referencing instructions 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 relocating loader resolves external symbol references and relocated the program units as described below.

(1) External symbol reference resolution

The relocating loader refers to the symbol table when resolving external symbol references (see Figure 7). 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 referenced.

When the loader 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 loader enters it into the symbol table as a new undefined symbol, loads the reference 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 loader loads the defined absolute address into the operand address in memory.
(3) If it has been cataloged but not defined, the loader moves the old reference address in the symbol table entry to the operand address in memory and loads the new reference address into the symbol table entry.

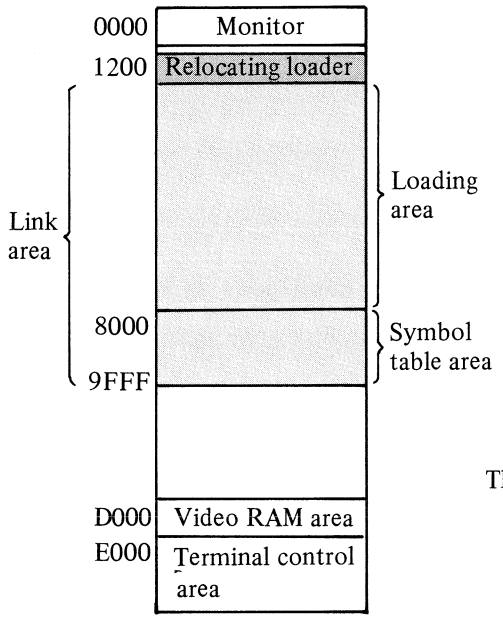
Thus, the loader chains undefined references to each symbol and, when the symbol is defined, replaces all reference addresses with the defined absolute address.

In other words, when an external symbol defined by the ENT assembler directive appears in the control frame, the loader 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 9).

When the first CALL SUB1 instruction is encountered in program unit 1, the loader enters SUB1 into the symbol table as an undefined symbol, loads the operand address (referenced 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 9 (a)).

When the CALL SUB1 instruction is encountered twice in program unit 2, the loader chains together their operand addresses which reference SUB1 (see Figure 9 (b)). When SUB1 is defined in program unit 3, the loader 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 9 (c) shows that SUB1 is defined by absolute address 5544H. When the loader subsequently encounters a CALL SUB 1 instruction, it immediately loads 5544H into the operand address of the instruction since symbol SUB 1 has been defined.



It is assumed that a 36K bytes (48K bytes maximum) of RAM are installed, and that the starting address of the symbol table is set to 8000H.

Fig. 7 Memory map for the relocating loader

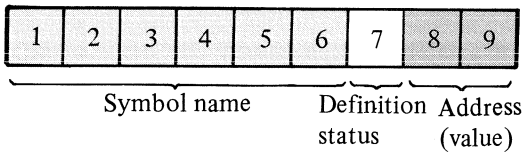
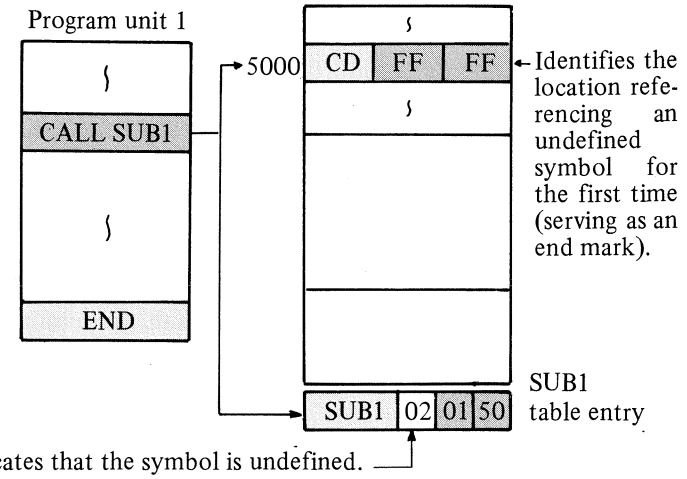
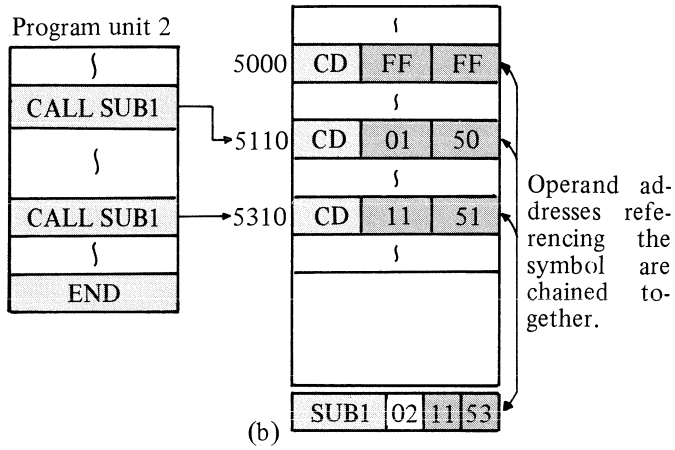


Fig. 8 Symbol table entry format





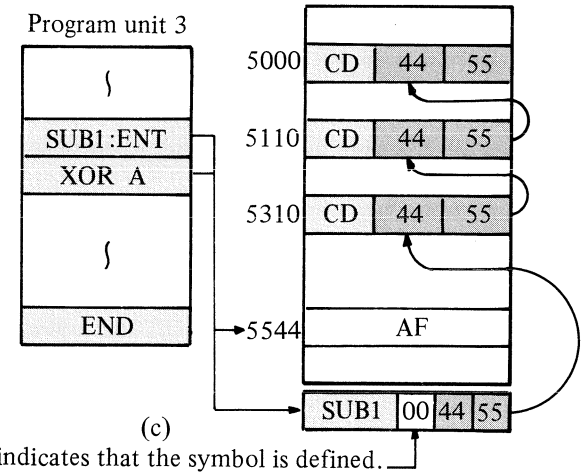


Fig. 9 Example of external symbol reference chaining

(2) Program relocation

The loader 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 loader produces absolute addresses for the symbols by adding bias to the relocation addresses.

Thus, the loader 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 loader 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 relocating loader 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 10).

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.

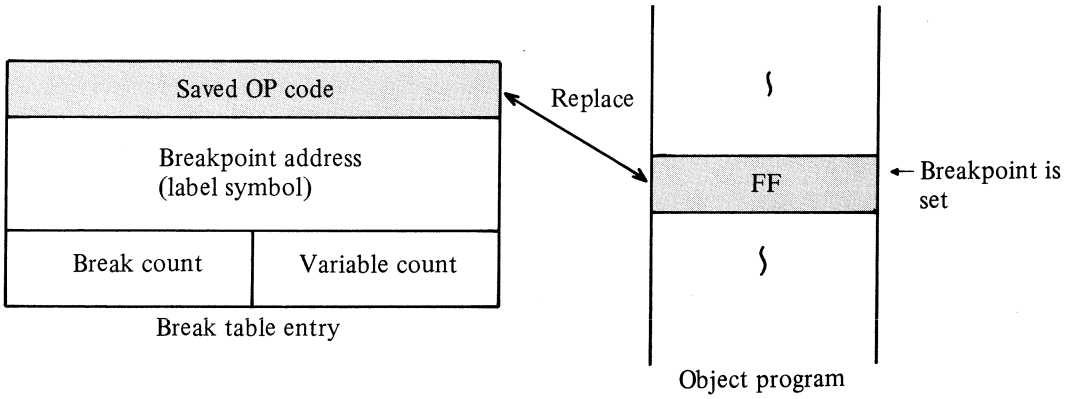
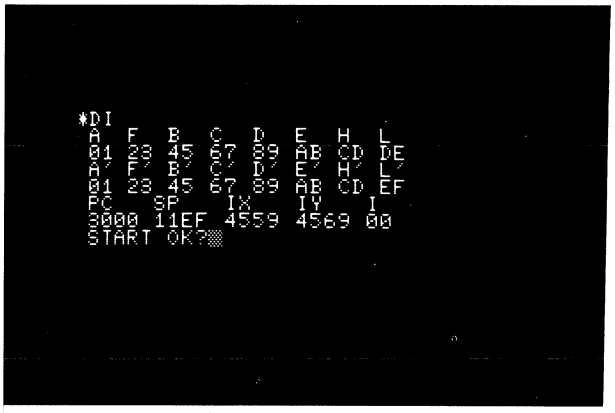


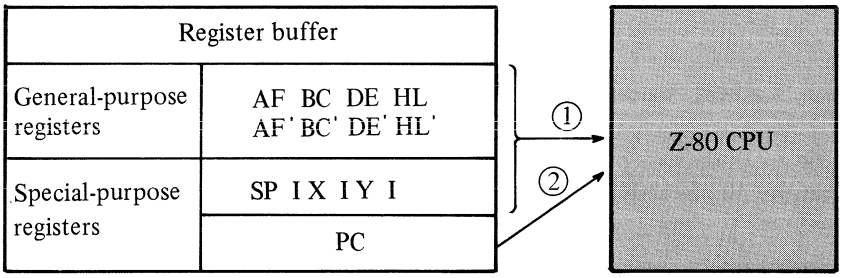
Fig. 10 Breakpoint setting and breakpoint table format

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 the 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 entered, the display shown in Figure 11 appears on the screen. When the programmer presses CR after confirming the CPU register contents, the debugger initiates an indicative starts as shown in Figure 12.



The above display shows that the program is to be started at address 3000 (hex) with the CPU register values shown.

Fig. 11 I command example



The debugger restores the contents of the general-purpose registers and special-purpose registers SP, IX, IY and I ①, then the value of the PC ② and initiates program execution.

Fig. 12 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 13. 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 99 lines away from the address represented by the symbol

(3) The address of an instruction 1 to FF (hexadecimal) lines away from the address represented by the symbol

(4) An address ±1 to 99 bytes away from the address represented by the symbol

(5) An address ±1 to FF (hexadecimal) bytes away from the address represented by the symbol of course, the programmer can also specify memory locations with absolute addresses.

For example, if the program unit whose source program is shown at the left of Figure 13 is loaded into memory by the debugger starting at hexadecimal address 3000, execution of a D command will display a dump of the memory block as shown at the right in Figure 13.

```
START  : ENT
         LD    SP, START
         CALL  MSTP
         XOR   A
         LD    (?TABP), A
         LD    B, A
MAINO  : ENT
         LD    A, OFH
```

Fig. 13 D command

2.5 PROM Formatter

The PROM formatter generates formatted absolute binary code and stores it into paper tape under PTP control. It is system backup software used to transfer object programs to the PROM writer. Currently, the following paper tape output formats are supported (see Figure 14):

(1) BNPF format: Brightronics, Intel and Takeda

(2) B10F format: Takeda

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

(4) Binary format: Brightronics

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

Fig. 14 Paper tape output formats

The PROM formatter is made up of a symbolic debugger and format, PTP and PTR controls (see Figure 15.) The programmer can perform debugging and format conversion simultaneously.

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 changed using the FC (parity Form Change) command.

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-80I/0-1). 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.

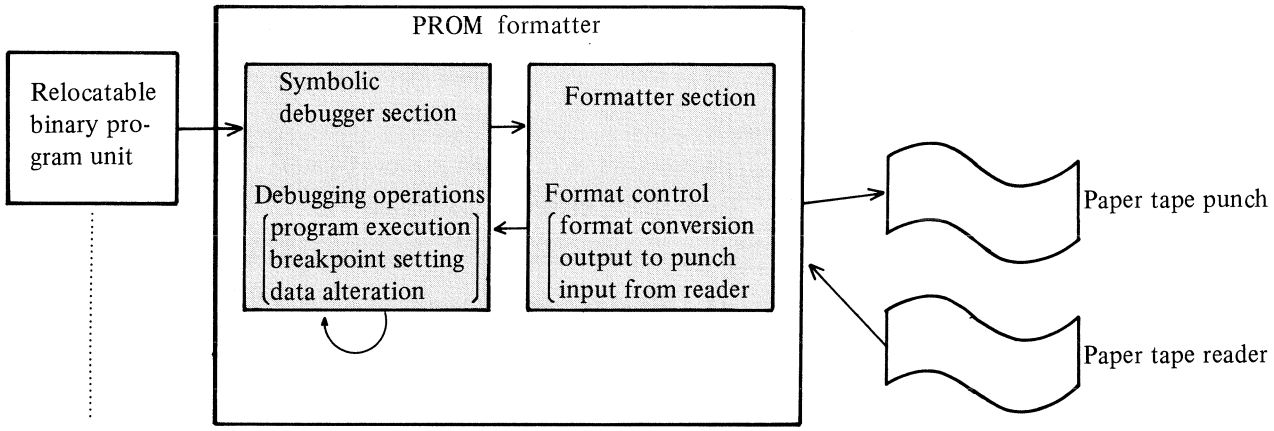


Fig. 15 PROM formatter configuration

3. FDOS ORGANIZATION

Figure 16 shows the files which are run under control of the SHARP MZ-80K 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

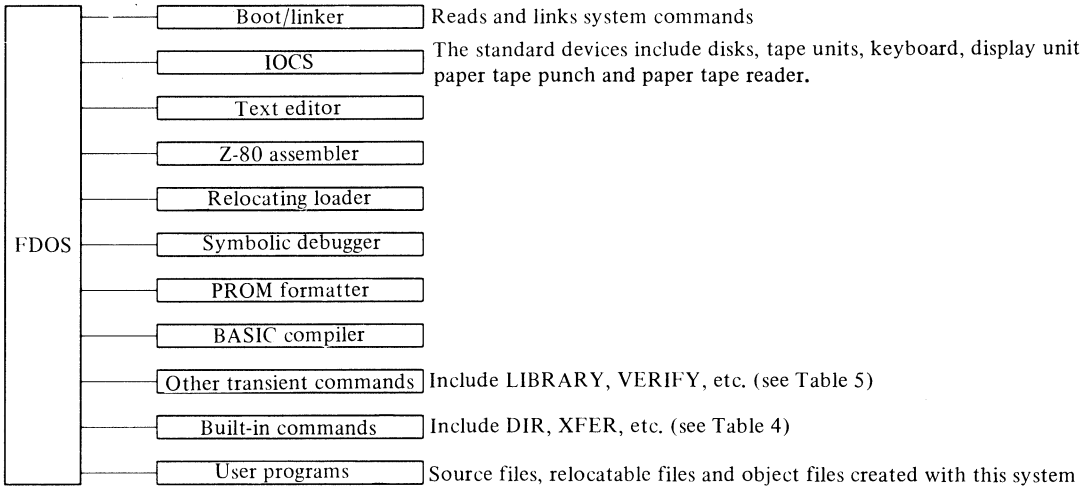


Fig. 16 FDOS file organization

Figure 17 shows the memory map for the above system resources. FDOS is made up of a resident section and an overlay section. The 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.

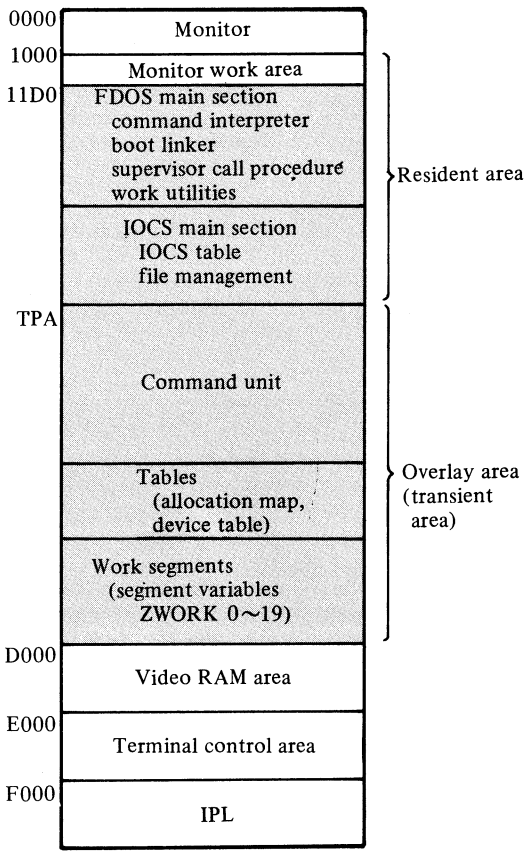


Fig. 17 FDOS memory map

The overlay area contains a command unit into which command programs are read by the boot linker. The work segment area is used to store 20 segment variables for dynamic segmentation and linkage.

3.1 File References

A file reference consists of a file name field and an extension field called the file mode as exemplified below.

ABCDEFGHIJKLMNOP.ASC

The file name must be 16 characters or less and the file mode must be three characters or less. The file mode is controlled by the FDOS main program. The legal file modes are listed in Table 3.

The following I/O device names are reserved by the FDOS main program:

$FD1 through $FD4:	Floppy disk units 1–4
$LPT:	Line printer
$CMT:	System cassette unit
$CRT:	System display unit
$KB:	System keyboard
$PTR:	Paper tape reader
$PTP:	Paper tape punch
$SIA:	Serial input port A
$SIB:	Serial input port B
$SOA:	Serial output port A
$SOB:	Serial output port B
$USR1 through $USR4:	User devices 1–4

Table 3 File modes

Mode	Meaning
.SYS	Relocatable binary system files under FDOS control
.ASC	Source files (ASCII files) created by the text editor
.RB	Relocatable files generated by the assembler or compiler
.OBJ	Object files
.LIB	Library files created by FDOS

Since the FDOS allows I/O devices to be treated as if they were separate files, the programmer can operate them in the same manner as logical files without switching running devices and giving detailed control commands.

Generally speaking, the file name and file mode must be explicitly defined. In particular, definition of I/O device files is mandatory. For convenience, the wildcard characters ? and * are provided for use when specifying groups of files. The symbol ? matches any character of a file name in the position in

which it is located. This is highly convenient when searching the directory for a file or when matching files. For example, A?C?.ASC is interpreted as explicitly defined file names ABCD.ASC, AXCY.ASC, etc.

The symbol * matches an arbitrary number of ? wildcard characters. Some examples are given below.

*.ASC	←→	ABCD.ASC
		????.ASC
XY*.RB	←→	XYZ.RB
		XYZ012.RB
ABC.*	←→	ABC.ASC
		ABC.RB

Commands and file names may be followed by switches. These are used to supplement or modify the command or file name. A switch must immediately follow a command or file name and be separated from it by a slash ("/"). Switches used with commands are called global switches and those used with file names, local switches. Examples of the use of switches are given in Figure 18.

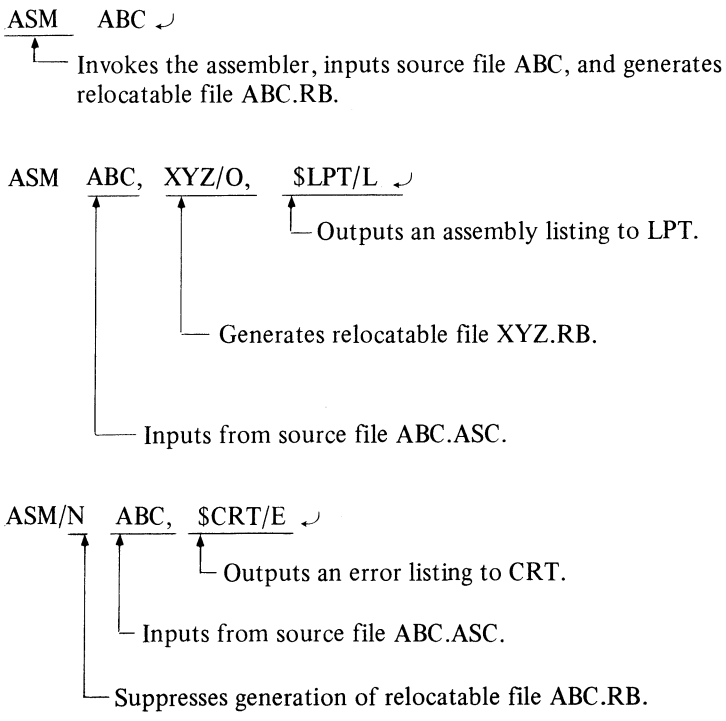


Fig. 18 Examples of switches

3.2 FDOS Command Summary

The FDOS commands are broadly divided into built-in commands (Table 4) and transient commands (Table 5).

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 Built-in commands

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.✳

TYPE filename 1 [, ..., filename N] (? ,✳)

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.

RENAME oldname 1, newname 1 [, ..., oldname N, newname N]

Renames the file specified by oldname to newname.

Examples:
- RENAME ABC, XYZ ↵ : ABC to XYZ.
- RENAME ABC, DEF, UVW, XYZ ↵ : ABC to DEF and UVW to XYZ.

DELETE filename 1 [, ..., filename N] (? ,✳)

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.
- ABC.ASC : DELETED : Indicates that the file is deleted since "Y" is entered.
- ABC. RB : Indicates that the file is not deleted since "N" is entered.
- AXY. OBJ : PERMANENT : Indicates that the file is not deleted because it is assigned the PERMANENT file attribute.

CHATR sign, filename 1, attribute [, ..., filename N. 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 attribute 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.

XFER sourcefile1, destinationfile 2 [,....., sourcefile N, destinationfile N] (?, ✳ only for source file)

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 PTR to file DEF.

XFER XYZ, $PTP/PE↵ : Transfers file XYZ to PTP with even parity in ASCII code.

RUN filename

Executes the program in the object file identified by filename. The file must be an OBJ mode file.

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

FREE [$FDn]

Lists statistical information about the disk identified by $FDn on the screen or on LPT.

Global switch/P : Outputs statistical information on 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.

DATA [MM/DD/YY]

Displays the current date or sets the specified data 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 LPT.

Examples: DATE/P↵ : Lists the current date on LPT.

DATE 12/25/80↵ : Sets the current date to December 25, 1980.

TIME [HH:MM:SS]

Displays the current time or sets specified time in hour, minute, second format. This information is used as file information when new files are created. 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 LPT.

Examples: TIME/P↵ : Lists the current time on LPT.

TIME 16 : 30 : 30↵ : Sets the current time to 16 : 30 : 30.

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.

HCOPY [message]

Lists the information currently displayed on the screen on LPT with the specified message.

Example: HCOPY↵

PAGE [output-device]

Performs a form feed operation on the output device identified by output-device.

Example: PAGE ↵ : Moves the printer form to the home position.

BYE

Terminates FDOS processing and returns control to the monitor.

Example: BYE↵

Table 5 Transient commands

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.

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 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 LPT.

LINK filename 1 [, ..., filename N]

Links relocatable files identified by filename 1 through filename N and outputs an object file with a link table listing.

Global switch/T: Specifies that the symbol table information it to be listed.

Global switch/P: Specifies that the listing is to be directed to LPT (the listing is displayed on the screen if the switch is omitted).

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 LPT.

DEBUG filename 1 [, ..., filename N]

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 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.

PROM

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

Example: PROM↵

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 relocatble file XYZ. RB.

LIBRARY filename 1 [, ..., filename N]

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 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.

VERIFY filename 1, filename 2 [, ..., filename N-1, filename N] (?,∗ only for filename 1 , ..., filename N-1)

Compares the contents of files filename 1 through filename N.

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

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

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.

COPY

Copies the files on the diskette designated by the source disk number to the diskette designated by the destination disk number. The system matches the passwords in these diskettes before carrying out a copy operation.

Example: COPY ↵

FORMAT [$FDn]

Initializes the diskette in $FDn to 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.

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).

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).

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.

LOAD filename

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

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

3.3 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 19).

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.

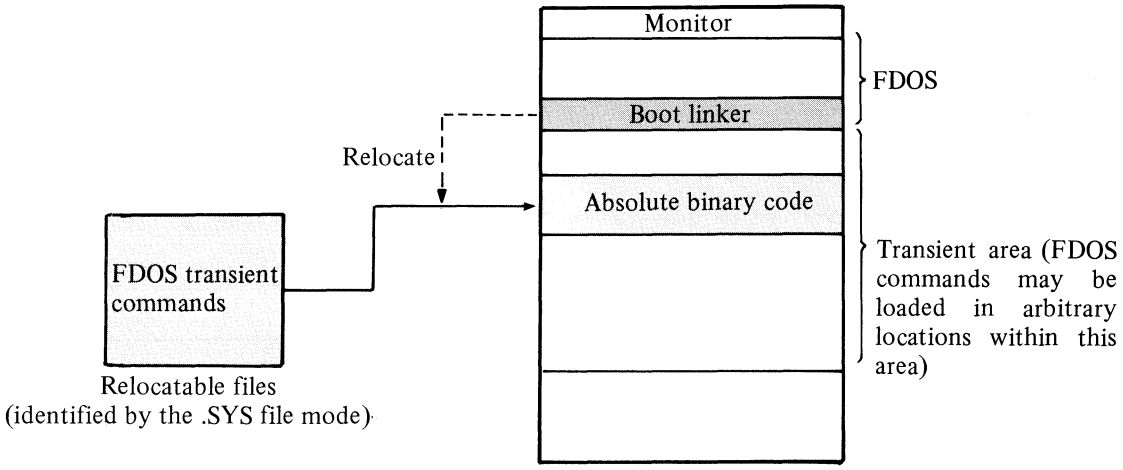


Fig. 19 Loading FDOS transient commands with the FDOS boot linker

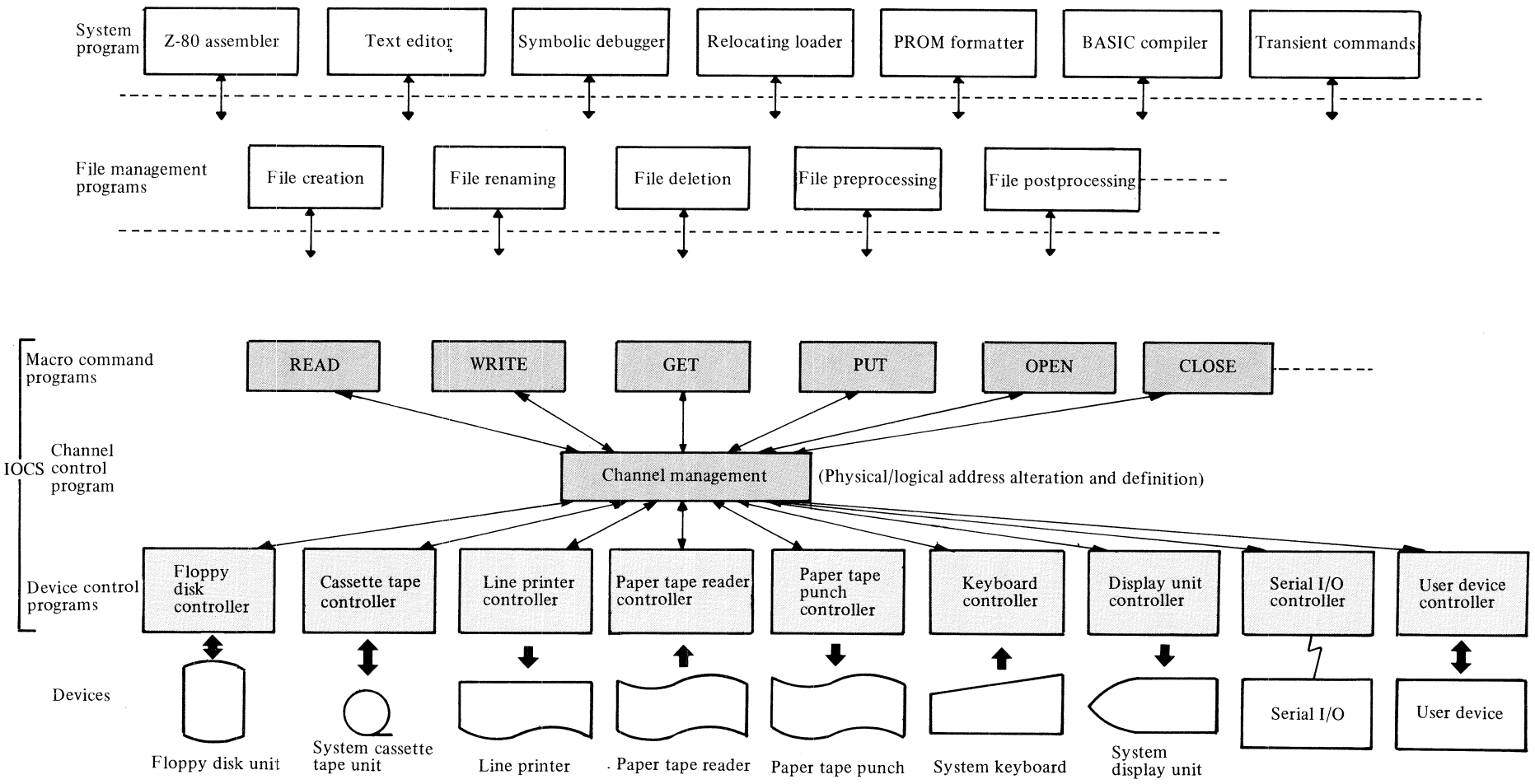
3.4 IOCS

IOCS in FDOS provides control over the display unit, cassette unit, floppy disk units, printer, PTP, PTR and other user I/O devices. 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 20. In the MZ-80K 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. 20 Hierarchical structure of file management programs



3.5 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 21 and 22. 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. 21 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

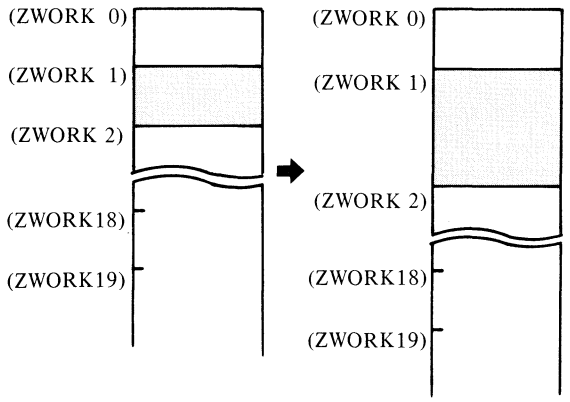
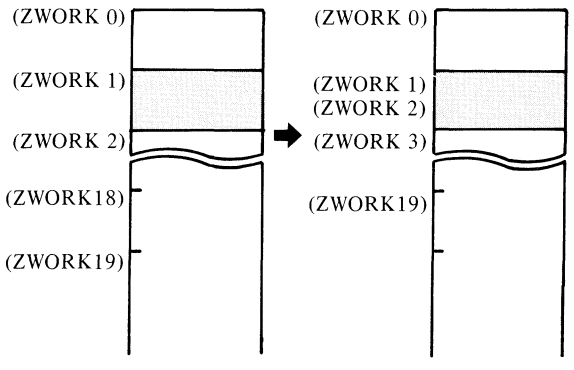


Fig. 22 Deleting a specified segment

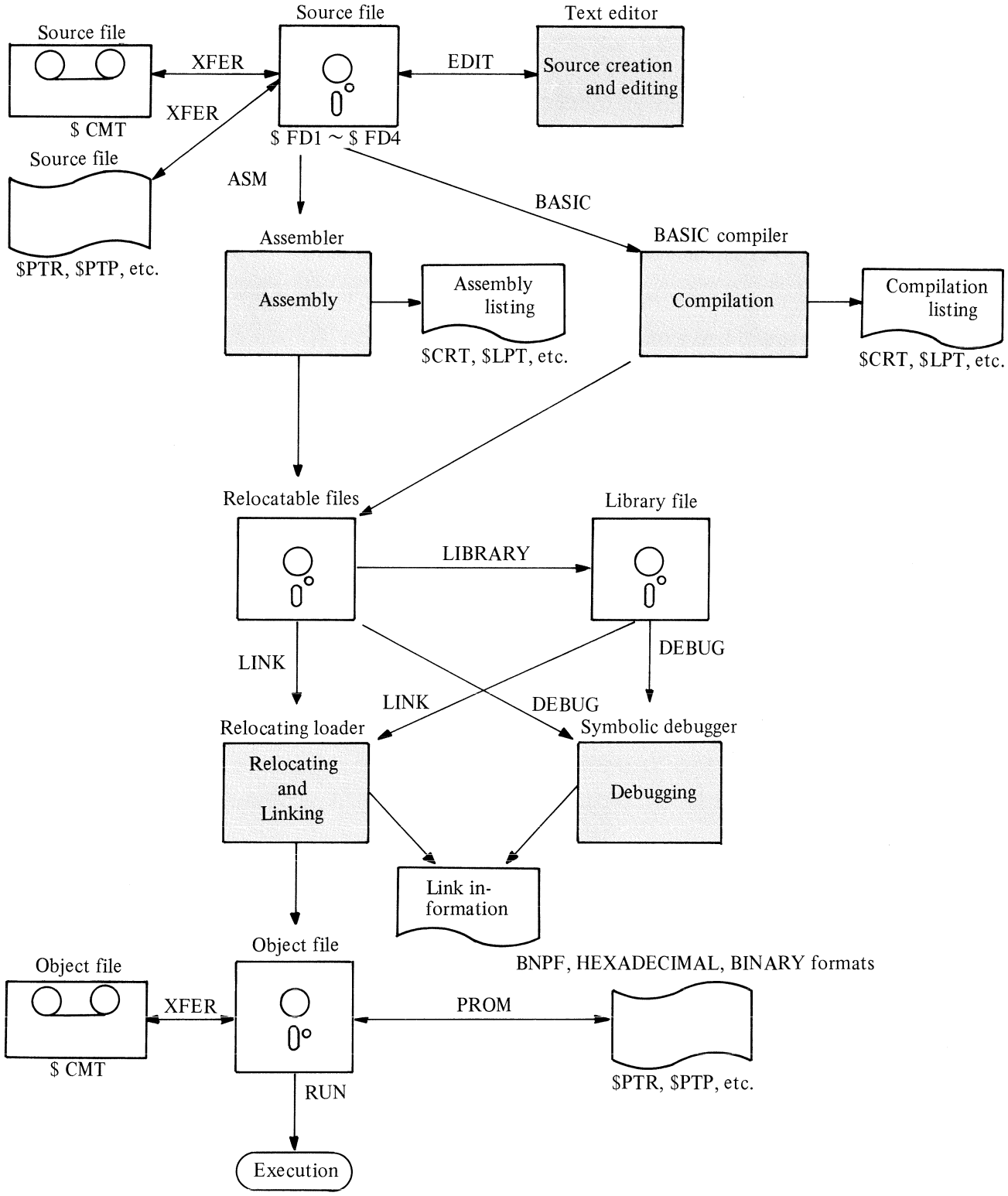
```
A←2          ; Segment No. (0–19)
BC←500       ; 500 bytes
CALL DDELET  ; DYNAMIC DELETE
```

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



4. FDOS COMMAND USAGE

4.1 Program Development Under FDOS



4.2 FDOS Command Coding Rules

This section describes the coding rules for FDOS commands.

— 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: 2 > EDIT␣TEST CR

2 — Default drive number (described later)
> — Prompt
EDIT — Command
TEST — Argument
␣ denotes a space.

The first number (1–4) indicated the default drive, namely, the currently logged-on disk drive. Some commands may require two or more arguments.

Example 2: 2 > XFER␣TEST, $CMT CR

XFER — Command
TEST — Argument 1
$CMT — Argument 2

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

(Legal) 2 > ␣XFER␣␣TEST␣$CMT CR

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

(Illegal) 2 > XF ER TEST , , $CMT CR

XF ER — No space is allowed.
, , — Only one comma is allowed.

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 three physical (screen) lines.

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

— 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 and/or special characters as shown below.

! # % & ' ( ) + – > = < @ [/] ↑ ←

No two files on a diskette can have the same file name and file mode (described later). Files with the same file name and mode may exist on different diskettes.

— **File modes** —

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

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.

— **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 #( )

— 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. An optional BASIC compiler is required to create, write to and read from random files.

— 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 PROG1.*	Deletes all files whose file name is PROG1.
XFER *.ASC, $PTP	Punches all files whose file mode is .ASC.
DELETE *.*	Deletes all files on the diskette.

– Drive number and volume number –

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

A volume number (1-255) is a number identifying a diskette. FDOS checks this number for validity each time it accesses a file.

– Device name –

FDOS can handle the following I/O devices:

$KB: MZ-80K system keyboard

$CRT: MZ-80K system display unit

$FD1:, $FD2:, $FD3:, $FD4: Floppy disk drives (MZ-80FD or MZ-80FDK)

$CMT: System cassette unit

$LPT: System printer (MZ-80P3)

$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.

$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. Refer to "Paper tape Punch and Reader Interface" in Appendix for details.

$SIA: Serial input port A
$SIB: Serial input port B
$SOA: Serial output port A
$SOB: Serial output port B

The interface card and control program for these I/O ports are options which will become available in the near future. The control program occupies addresses $C800 to $CFFF.

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

Notes

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

   1 > XFER $KB, XYZ

   is terminated when the programmer presses the SHIFT and BREAK keys simultaneously.
2. The auto repeat interval can be changed by the STATUS $KB, $00nn command (standard nn value is 10).
3. 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).
4. $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.
5. $CMT can handle only .ASC and .OBJ mode files. $KB, $CRT, $LPT, $PTR and $PTP 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).
6. $PTP and $PTR automatically skip the tape feed portions.

–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 5. In such cases they may be placed in any order.

Example 4: `1 > DATE/P` — /P denotes LPT.

(Command: DATE; Global switch: /P)

Example 5: `1 > LINK/P/T␣ TEST` — /P denotes LPT.

/T denotes the symbol table.

(Global switch: /P/T)

Invalid: `1 > LINK ␣ /P ␣ / ␣ T ␣ TEST`

(No space may appear in these positions.)

Local switches

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

Example 6: 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, $SIA, $SIB, $SOA and $SOB.

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

/P: Global switch; /O: Local switch

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

–Default assumptions–

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

[Device name] ; [File name] . [File mode]

Example 8: $FD2 ; PROG2 .ASC $CMT ; TEST2 .OBJ

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

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 9: 2 > LINK TEST1, $FD3; TEST2, TEST3

In the above example, the system assumes the name of the currently logged-on disk (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 10), the system assumes an appropriate file name.

Example 10: XFER $FD1 ; ABC.ASC, $FD2

└The system assumes $FD2 ; ABC.ASC.

Default file mode

When 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.

–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 (default file mode)
   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: TIME 9 : 30 : 00
   ↑Command ↑Argument

   LIMIT MAX
   ↑Command ↑Argument

–Suspending or stopping program execution –

Execution of an FDOS command may be suspended or terminated with the following keys:

[SHIFT] + [BREAK] The system terminates the current program, issues the message "BREAK" and waits for entry of an FDOS command. This break can be detected by the "ON BRKEY GOTO" statement of BASIC.

[SP] When the [SP] key is held down for a while, the system suspends execution of the current program. The phrase "for a while" here refers to the period of time until a carriage return is output during output to the CRT screen or printer. At this point, the programmer can take one of the following actions:

- [SHIFT] + [BREAK] The system reacts as described above.
- [SP] Processing is continued at the point of the suspension.

4.3 Using FDOS Commands

–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; TEST 1/L, $FD2;TEST 1. RB/O

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

Format

ASSIGN devicename 1, $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) ASSIGN $USR1, $C000

Assigns device name $US1 to the user I/O control rouitne 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.

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; .ASC 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 code 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

1. The compiler terminates generation of the relocatable file when it detects an error during compilation.
2. The BASIC compiler is available as an option.

— BYE —

Built-in

Format

BYE

Function

The BYE command returns control to the monitor.

— 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.

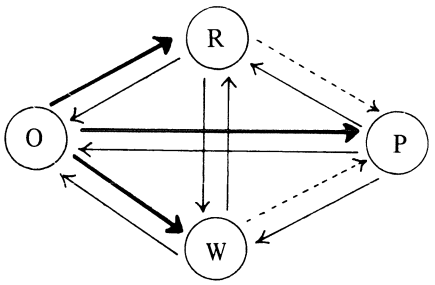
(2) **CHATR SECRET, TEST. OBJ, 0**
Deletes the file attributes of file TEST .OBJ. The specified password, "SECRET", is matched with the password specified for the file before the command is actually executed.

(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.

Format

CONVBTX filename

Function

The CONVBTX command copies BASIC text files generated with the SP-5000 series BASIC interpreter or D-BASIC SP-6015 onto a diskette in the FDOS compatible format.

The BASIC text files are input from the cassette tape unit.

Default file mode

.ASC

Switches

None.

Wildcard characters

Not allowed.

Example

(1) CONVBTX TEST

↓ PLAY

FOUND XYZ.BTX

LOADING XYZ.BTX

Reads a BASIC text file from the cassette tape unit and creates file TEST .ASC in the ASCII format.

Programming notes

1. To use a file created and stored with D-BASIC under FDOS, save the text file on cassette tape through D-BASIC, then execute the FDOS CONVBTX command.
   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 CONVBTX 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.

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 pass-
                                          words match.
INSERT SOURCE INTO  $FD 1      ←Insert the source diskette in drive FD1.
   DESTINATION INTO  $FD 2,    ↓SPACE KEY ←Insert the destination diskette in drive
                                          FD2, then press the SP key.
2 >                                       Copying is completed.
```

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 11/20/81

Sets the system calender date to November 20th, 1981.

(2) DATE

Displays the current date on CRT.

(3) DATE/P

Prints the current date on LPT.

Format

DEBUG filename1, ..., filenameN

Function

The DEBUG command links and loads relocatable files specified by the arguments to form an object program into 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.

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.

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 pseudo-file $MEM.

Format

DIR devicename (filename)

Function

Displays the contents of the directory specified by devicename or 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 default drive 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

— 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.

— 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 commands 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, FDOS.LIB
CHATR KEY, $FD2;TEST. OBJ. W
RUN $FD2; TEST
3 > FREE          Display the number of used sectors on the diskette in FD3.
DIR/P $FD2        Print the contents of the FD2 directory on LPT and designate FD2 as the
                  current default drive.
```

(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.
ASM TEST , $USR1/L             Assemble relocatable file TEST and output the assembly listing
                               on the user-defined printer.
XFER $USR2, XYZ                Transfer data obtained with a light pen to file XYZ.
```

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 Ror 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.

— 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 SHIFT and BREAK keys simultaneously to return to FDOS.

The above interaction shows an example of formatting a completely new diskette. "SIGN" prompts for a password to be given to 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 255. 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. ? 128
END
INSERT DISKETTE INTO $FD1, ⬇ SPACE KEY
BREAK ←— Press the SHIFT and BREAK keys simultaneously 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).

— 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 : 128 LEFT : 1056 USED : 64

(2) FREE/P

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

Programming note

A diskette comprises 1120 sectors (each consisting of 128 bytes). Of 1120 sectors, however, 64 sectors are reserved by the system as FDOS areas. Consequently, USED: 64 is indicated for new diskettes.

— 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 ←— The [INST] key is pressed with the cursor over the character ■ since double quotation marks ("") would be printed as is.

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 ":".

HCOPY "abcde" ←——Not allowed

(2) The following are LPT mode control codes.

■ Paging: Feeds the paper to the position where power has been turned on.

■ Suppressed spacing: Used for graphic display, etc.

■ Double size characters: Used for titles, etc.

■ Clear: Clears the ■ and ■ functions.

and ■, ■ are ignored.

— 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.

— 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

— LINK — Transient

Format

LINK filename1, ..., filenameN

Function

The LINK 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 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.

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.

Wildcard characters

Not allowed.

Examples

(1) LINK TEST1, TEST2

Links relocatable files TEST1.RB and TEST2.RB and generates an object file named TEST.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, FDOS.LIB, EXEC $C100

Links TEST.RB and FDOS.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).

— 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.

— PAGE — Built-in

Format

PAGE output-device

Function

The PAGE command carries out a paging operation on the output device specified by output-device.

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, $SOA, $USR1

Produces a feeder tape on PTP and outputs the code defined with the STATUS command to SOA and USR1.

— 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.

— RENAME — Built-in

Format

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

Function

The RENAME command renames specified files.

Default file mode

.ASC

Switches

None.

Wildcard characters

None.

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.

— RUN — Built-in

Format

Run filename

Function

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

Default file mode

.OBJ

Switches

None for normal use. Using switches, see the explanations under "Linking Assembly Programs with FDOS" in Appendix and "FDOS Subroutines" in Library/Package.

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 presses 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.

— 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 serial nunber as 79.

— 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 Coded I/O Routines "in Appendix for details.

Default file mode

None.

Switches

None.

Wildcard characters

None.

Examples

(1) STATUS $SOA, $1234

Sets the SOA control status to 1234 (hexadecimal).

(2) STATUS $USR1

Displays the control status of USR1 on CRT.

Programming note

This command is available for the serial I/O devices ($SIA, $SIB, $SOA and $SOB), and user devices ($USR1 to $USR4).

— 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

Set 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.

— TYPE — Built-in

Format

TYPE filename1, ..., filenameN

Function

The TYPE command outputs the 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.

— VERIFY — Transient

Format

VERIFY sourcefile1, destinationfile1, ..., 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 with that name on the diskette in FD2. Note that only the first file whose file name matches TEST* is taken.

— XFER — Built-in

Format

XFER sourcefile1, destinationfile1, ..., sourcefileN, destinationfileN

Function

The XFER command transfers the contents of the source files to the 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 dikette in FD2. An error is generated if file on CMT has no file name.

(5) XFER $CMT;TEST*, $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 fiel TEST.ASC. The file read from the keyboard is terminated by pressing the SHIFT and BREAK keys simultaneously.

(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 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.

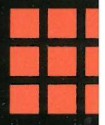
The system error messages are listed below.

```
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	
54	NOT OPENED	
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 on a diskette exceeds 96, or opened too many files simultaneously.
62	DISK VOLUME	; Diskette replaced improperly.
63	NO FILE SPACE	; Diskette has no free space.
64	UNFORMAT	; Diskette unformatted.
65	FD HARD ERROR	; Hardware related disk error
66	IL DATA	
67	NO USABLE DISKETTE	
68	(SUB) MASTER DISKETTE	; Master, submaster and/or slave diskette is misused.
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	
81	FLAMING	} Serial I/O errors (to be implemented later)
82	OVER RUN	
83	INTERCONNECT	
84	FULL BUFFER	
85	UNCONTROLLABLE	} IEEE-488 related errors (to be implemented later)
86	INTERFACE	
87	LESS DATA	
88	MUCH DATA	
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.

Text Editor



SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

— CONTENTS —

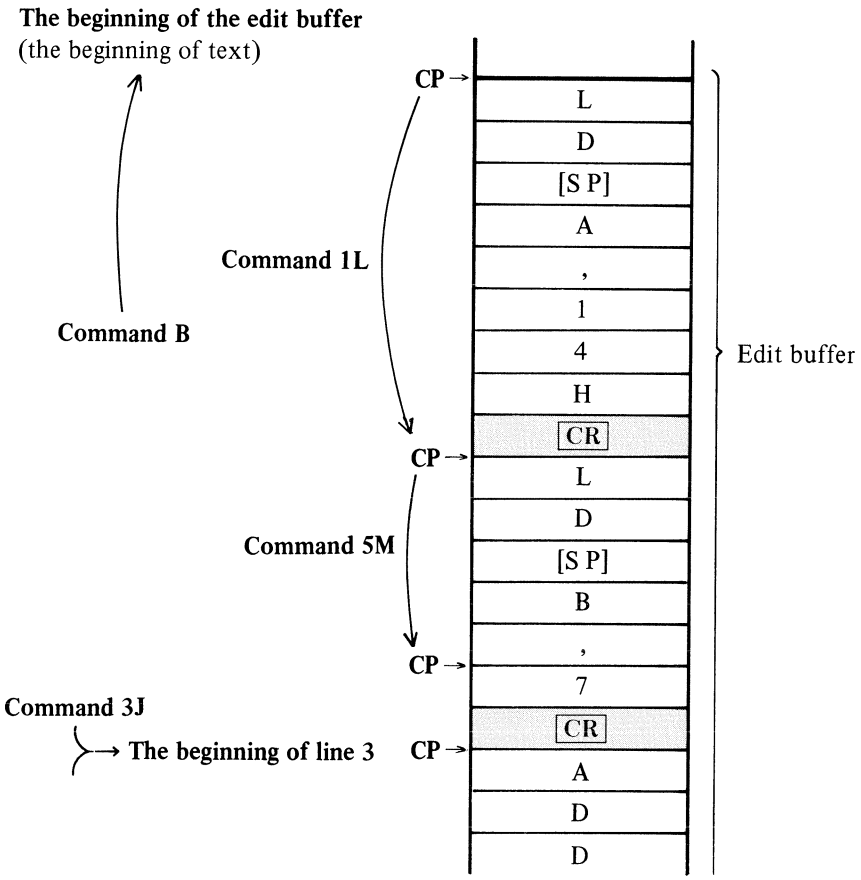
CHARACTER POINTER AND DELIMITER

The character pointer (CP) is positioned **at the boundary between two adjacent characters** or the beginning or end of the text. It does not point directly at any character.

Movement of the CP is explained below based on the assumption that the following text is stored in the edit buffer.

```
1 LD A, 14H
2 LD B, 7
3 ADD A, B
4 DAA
```

(Line numbers are not stored in the edit buffer.)



The B command moves the CP to the beginning of the edit buffer, the J command to the top of the specified line and the L command to the beginning of the n lines from the line at which it is currently located; the top of the specified line is the boundary following the [CR] code of a preceding line.

Delimiters are used as separators between editor commands. Entering several editor commands and separating them with delimiters allows them to be executed consecutively by pressing the [CR] key.

The I (Insert) command must be followed by a delimiter because it uses [CR] codes as character codes for the source text. The following example replaces ADD on line 3 in the above program with ADC.

3J→2M→1D→IC→ [CR] or **B→CADD→ADC** [CR]

TEXT EDITOR COMMANDS

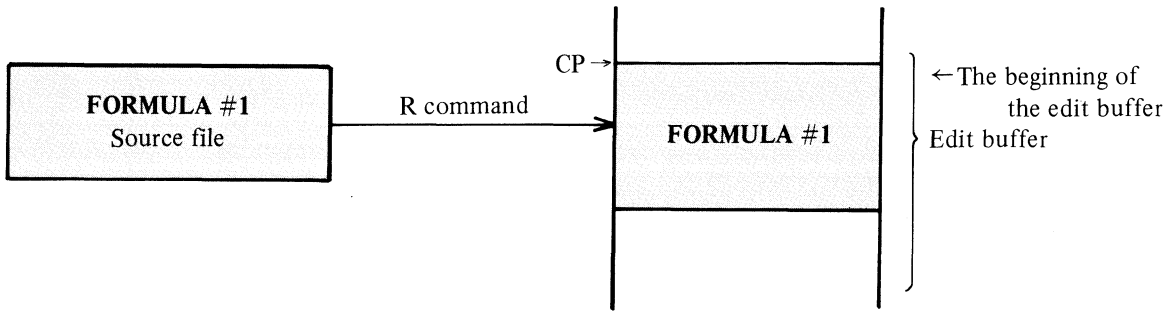
——INPUT COMMANDS——

R (Read file) Command

This command clears the edit buffer area, then loads it with the source file (ASCII file) specified by the filename; loading starts at the beginning of the edit buffer. The CP is positioned at the beginning of the edit buffer after execution of this command.

* **RFORMULA #1** CR	Reads file **FORMULA#1** into the edit buffer.

— Enter R followed by the file name while in the command wait state.
— The editor searches for the file and reads it.
— The file is stored starting at the beginning of the edit buffer as shown below.
— The CP is positioned to the beginning of the edit buffer after reading.



— The message "FULL BUFFER" is displayed when the edit buffer becomes full. In this case, only part of the input file is stored in the edit buffer.

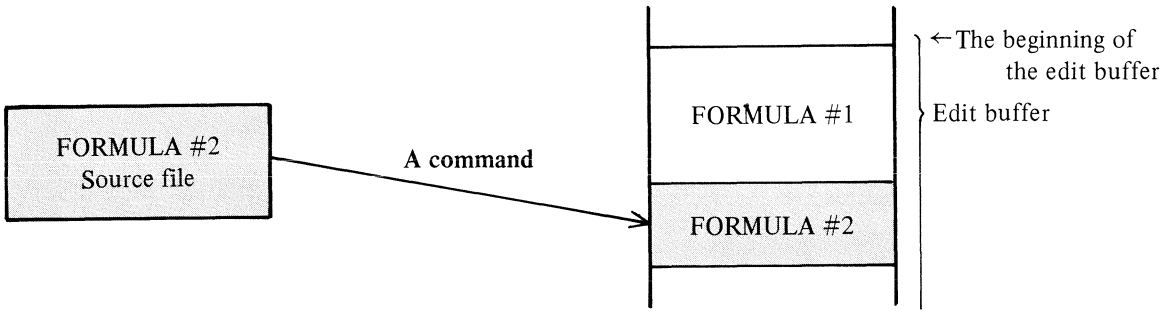
— "*" is displayed to indicate that the system is the command wait state.

A (Append file) Command

This command appends the file specified by the filename to the contents of the edit buffer. The CP position is not changed.

＊ **AFORMULA # 2** CR	Appends the source file **FORMULA#2** to the contents of the edit buffer.

— Enter A followed by the file name while in the command wait state.
— The editor searches for the specified file and reads it.
— The file is stored in the area following the end of the last text in the edit buffer. The figure below shows a case in which the file FORMULA#2 is appended to the file FORMULA#1.



— The message "FULL BUFFER" is displayed when the edit buffer becomes full. In this case, only part of the specified file is stored in the edit buffer and the contents of the edit buffer must be reedited to store the entire file.

——OUTPUT COMMAND——

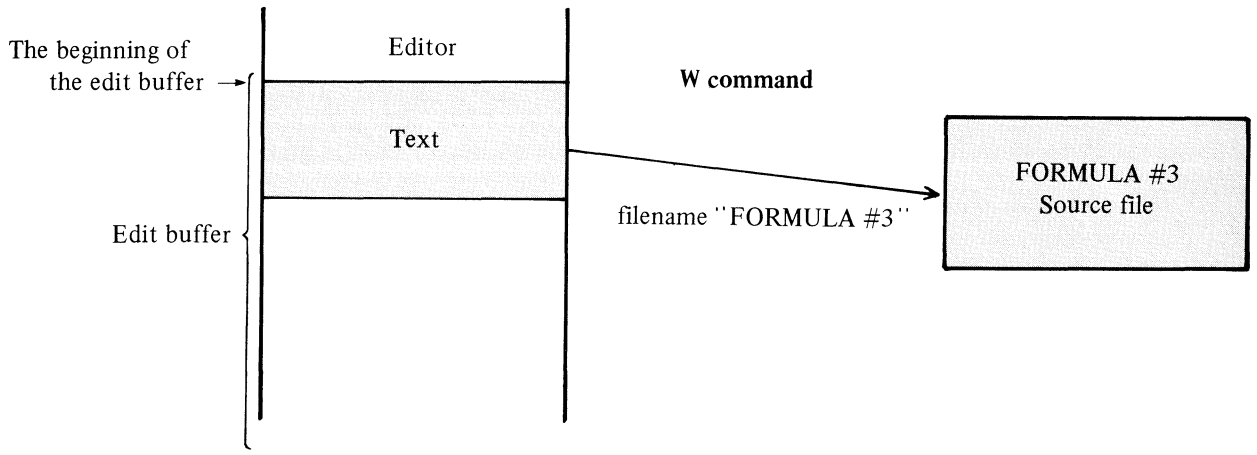
W (Write) Command

This command outputs the entire contents of the edit buffer to the file specified by the filename regardless of the CP position.

＊WFORMULA#3 CR	Outputs the contents of the edit buffer to file FORMULA#3 in the active drive.
＊W$FD2;FORMULA#3 CR	Outputs the contents of the edit buffer to file FORMULA#3 in floppy disk drive 2 ($FD2).

— Enter W followed by the file name while in the command wait state.

— The editor waits for entry of another command after the edit buffer contents have been output. The file generated is a source file.



— The CP position is not changed by execution of the W command.

——PAGE PROCESSING COMMANDS——

These commands are used in cases where the total size of files to be edited exceeds the size of the edit buffer, as shown in the following examples.

If the diskette is replaced with a new one during page processing, the contents of the diskette may be destroyed. Be sure to terminate page processing before replacing the diskette.

1. When the file to be edited is larger than the edit buffer:

❶ ✳**PRABC** [CR] Reads file "**ABC**" into the edit buffer until the buffer is full.

Note: Omission of the file name in the first page processing command will result in an error.

❷ ✳**PWDEF** [CR] Outputs the contents of the edit buffer to file "DEF" after editing is completed.

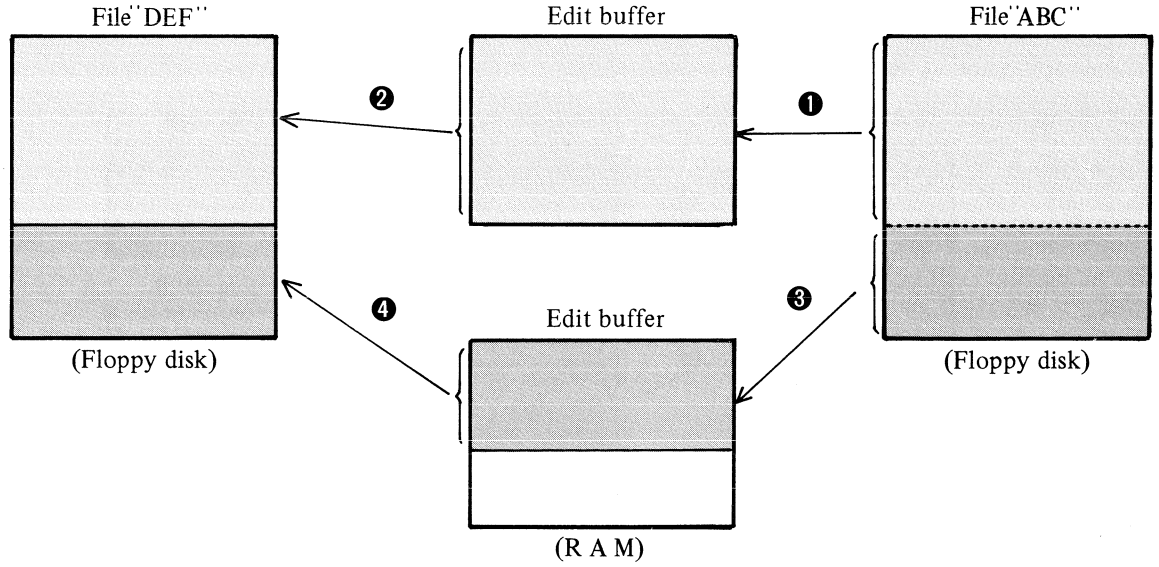
Note: ✳**PW** [CR] Results in an error. An error results when editing increases the size of file in the edit buffer so that it exceeds the size of the edit buffer.

❸ ✳**PR** [CR] Reads the remainder of a file specified by a preceding PR command into the edit buffer. In this example, the command reads the remainder of file "**ABC**" into the edit buffer.

Note: A file name remains valid after it is specified in a PR command until a new file is specified.

❹ ✳**PW** [CR] Outputs the contents of the edit buffer and appends it to the file specified by the preceding PW command after editing is completed. In this example, the command appends the contents of the edit buffer to file "**DEF**".

❺ ✳**PC** [CR] Terminates page processing. (This command is mandatory).



2. When the file to be edited first is larger than the edit buffer and another file is to be edited and appended to the first edited file:

❶ ✳PRABC [CR] Reads file "ABC" into the edit buffer until the buffer is full.

❷ ✳PWDEF [CR] Outputs the contents of the edit buffer to file "DEF" after editing is completed.

❸ ✳PR [CR] Reads the remainder of file "ABC" into the edit buffer.

❹ ✳PW [CR] Appends the contents of the edit buffer to file "DEF" after editing is completed.

❺ ✳PRGHI [CR] Reads file "GHI" into the edit buffer.

Note: If this case, specifying ✳PR [CR] will not be valid if the end of file "ABC" has been reached.

❻ ✳PW [CR] Appends the contents of the edit buffer to file DEF" after editing is completed.

❼ ✳PC [CR] Terminates page processing. (This command is mandatory).

3. When several files are to be edited and the resulting file is larger than edit buffer:

❶ ✳PRABC [CR] Reads file "ABC" into the edit buffer.

❷ ✳PADEF [CR] Reads file "DEF" and appends it to the contents of the edit buffer until the buffer is full.

Note: If ✳PRDEF [CR] is entered, the contents of file "ABC" stored in the edit buffer are cleared and file "DEF" is loaded in the edit buffer from its beginning.

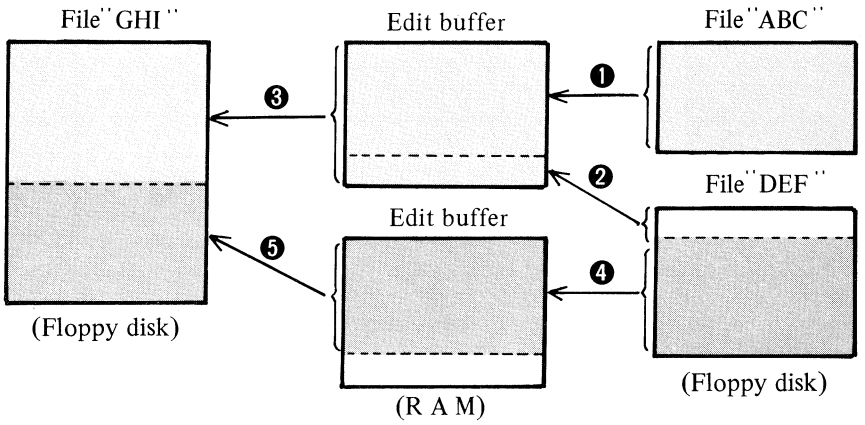
❸ ✳PWGHI [CR] Outputs the contents of the edit buffer to file "GHI" after editing is completed.

❹ ✳PR [CR] Reads the remainder of file "DEF" into the edit buffer.

Note: The input file was changed at step 2 ("ABC" → "DEF").

❺ ✳PW [CR] Appends the contents of the edit buffer to file "GHI" after editing is completed.

❻ ✳PC [CR] Terminates page processing.



——TYPE COMMAND——

T (Type) Command

This command displays all or part of the contents of the edit buffer. The CP position is not changed.

＊T CR	Displays all of the contents of the edit buffer with line numbers attached.
＊nT CR	Displays the **n** lines following the **CP**. (Same as the above when **n = 0**.)

— Key in the number of lines, n followed by T (Type) while in the command wait state.

— Press CR to display the contents of the edit buffer.

— The following are special cases of nT.

n = 0 : the same as T

n < 0 : results in the error message "???".

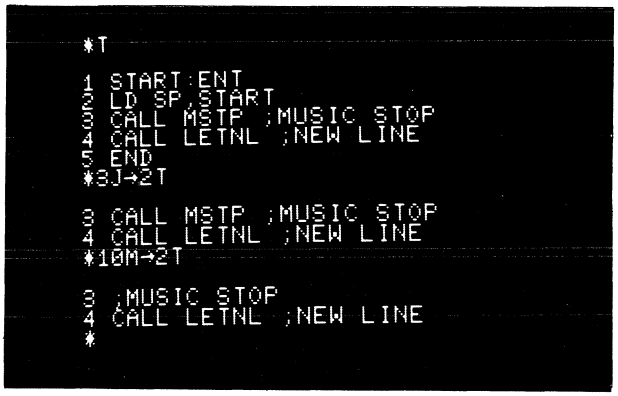
n ≧ m, where m is the number of lines from the one at which the CP is located to the end of the buffer contents: only m lines are displayed.

— The current CP position can be determined with the nT command, since display starts with the character following the boundary at which the CP is located.

— Press the SHIFT and BREAK keys simultaneously to terminate T command execution. Press the SPACE kay to suspend T command execution, and press it again to resume it.

— The photograph at right shows the relationship between the type command and the CP for the following text.

```
1 START : ENT
2 LD  SP, START
3 CALL  MSTP; MUSIC STOP
4 CALL  LETNL; NEW LINE
5 END
```



— The error message "LARGE" is displayed when n exceeds 65535.

——CP POSITIONING COMMADS——

B (Begin) Command

＊**B** CR	Positions the **CP** to the beginning of the edit buffer.

— Key in **B** while in the command wait state.
— Press CR
— The B command is executed to position the CP to the beginning of the edit buffer.

— nB performs the same function.

Z Command

＊**Z** CR	Positions the **CP** to the end of the contents of the edit buffer.

— Key in **Z** while in the command wait state.
— Press CR
— The Z command is executed to position the CP to the end of the contents of the edit buffer.

— nZ performs the same function.

J (Jump) Command

＊**nJ** CR	Positions the **CP** to the beginning of line **n**.

— Key in **line number n and J** while in the command wait state.
— Press CR
— The nJ command is executed to position the CP to the biginning of line n.

— The following are special cases.

n = 0 or 1 or n is omitted: the command performs the same function as the B command.

$n < 0$: results in the error message "???".

$n \geq m$ (where m is the number of lines of the edit buffer contents): the command performs the same function as the Z command.

L (Line) Command

This command moves the CP forward or backward the specified number of lines. The CP is positioned at the beginning of the specified line after execution.

∗nL CR	Moves the **CP** to the beginning of the **nth** line from the line at which it is currently located.
∗L CR	Moves the **CP** to the beginning of the line at which it is currently located.

— Key in number of lines, **n and L** while in the command wait state.
— Press CR .
— The CP is positioned at the beginning of the specified line when the nL command is executed.

— The following are special cases:

n = 0 : the command functions in the same manner as the L command.
n ≧ m (where m is the number of lines from the line on which the CP is located to the end of the edit buffer contents) : the command functions in the same manner as the Z command.
n < 0 : the CP is moved | n | lines toward the beginning of the edit buffer.
| n | ≧ ℓ − 1 (where ℓ is the number of the line at which the CP is currently located) : the command functions in the same manner as the B command.

M (Move) Command

This command moves the CP forward or backward by the specified number of characters. Spaces and carriage returns are counted as characters, but line numbers are not.

∗nM CR	Moves the **CP** to the position which is **n** characters from its current position.

— Key in number of characters, **n and M** while in the command wait state.
— Press CR .
— Executing the nM command moves the CP to the specified boundary between characters.

— When n < 0, the CP is moved backward by | n | characters.
— The CP position is not changed when n = 0 or if it is omitted.

——CORRECTION COMMANDS——

C (Change) Command

This command replaces a string in the edit buffer with another string. The search for the specified string starts at the current CP position and proceeds toward the end of the edit buffer; the string is replaced when it is found and the CP is positioned at the end of the string replaced.

✳Cstring 1→ string 2 CR	Searches for the character string specified by **string 1**, starting at the current **CP** position and proceeding toward the end of the edit buffer; replaces the string with the one specified by **string 2** when it is found.
✳Cstring 1 CR	Deletes the character string specified by **string 1**.

— Key in **C** while in the command wait state.
— Key in the string to be located followed by a delimiter.
— Key in the string which is to replace the one located.
— Press CR and a search is made for the first string. Only the first occurrence of the string is replaced. The line including the string replaced is displayed and the CP is positioned at the end of that string.
— The message "NOT FOUND" is displayed if the specified string is not found and the CP is positioned to the beginning of the edit buffer.

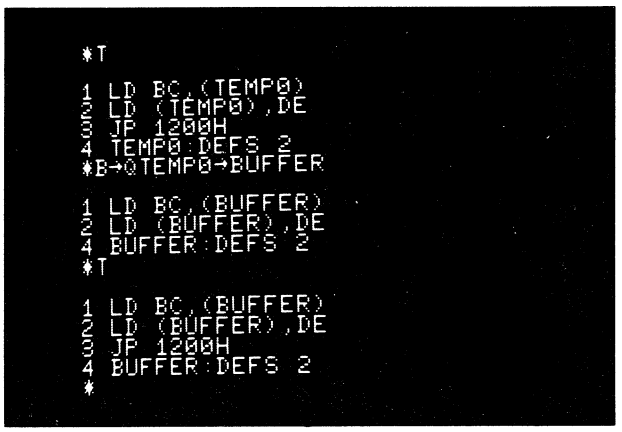
Q (Queue) Command

This command repeates the function of the C command each time the specified character string is found until the end of the edit buffer is reached. The CP is repositioned to the end of the string last replaced.

✳Qstring 1→ string 2 CR	Causes the function of the **C** command to be executed repeatedly.
✳Qstring 1 CR	Deletes all occurrences of the character stiring specified by **string 1**.

— Key in **Q** while in the command wait state.
— The remainder of the operation is the same as for the C command.
— The photograph at right shows the result of execution of the Q command on the following text.

```
1 LD BC,(TEMPO)
2 LD (TEMPO),DE
3 JP 1200H
4 TEMPO:DEFS 2
```



I (Insert) Command

This command inserts the specified string at the CP position. A carriage return is performed on the CRT screen if one is included in the string.

Line numbers are updated automatically when a new line is inserted. The CP is repositioned to the end of the string inserted.

✳**Istring** → [CR]	Inserts the specified **string** at the **CP** position.
✳**Istring1** [CR]	Inserts the lines specified by **string 1**, **string 2** and **string 3** at the **CP** position.
string2 [CR]	
string3 [CR]	
→[CR]	A [CR] is treated as a character by the I command. Therefore, a delimiter must be keyed in before [CR] is pressed to execute the command.

— Key in **I** while in the command wait state.

— Key in the string to be inserted.

— Strings keyed in are inserted at the CP position and the contents of the edit buffer following the CP are automatically shifted toward the end of the edit buffer.

— When a [CR] is keyed in, it is inserted as a carriage return code.

— Key is a delimiter (→) after all the strings have been keyed in.

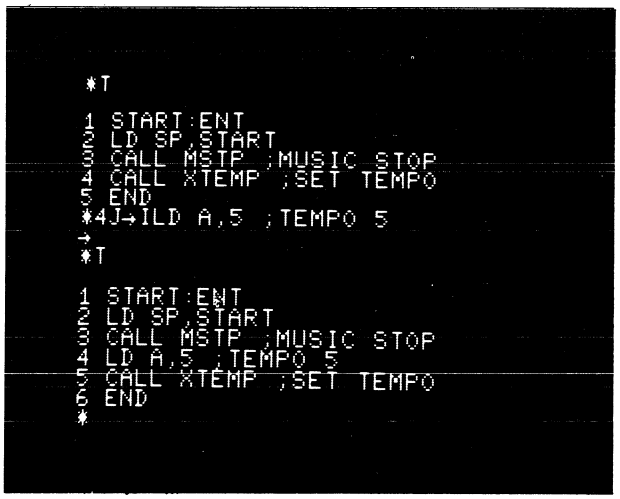
— Press [CR] key to execute the I command.

— The photograph at right shows an example of using the I command.

Text:

```
1 START : ENT
2 LD SP, START
3 CALL MSTP;MUSIC STOP
4 CALL XTEMP;SET TEMPO
5 END
```

LD A, 5 ;TEMPO 5 is inserted between lines 3 and 4 of the above text.



K (Kill) Command

This command deletes the n lines preceding or following the CP from the edit buffer.

✳**nK** CR	Deletes the **n** lines preceding or following the **CP** from the edit buffer. A line is not deleted if the **CP** is located within it, since characters preceding or following the **CP** are not deleted.
✳**K** CR	Deletes characters preceding the **CP** position until a **CP** is detected. The CR is not deleted.

— Key in the number of lines, **n and K** while in the command wait state.

— Press CR to execute the K command.

— Operation differs according to the value of n as follows.

n > 0 : Deletes all characters following the CP until n CR codes are detected. CR codes detected are also deleted. Command execution ends after the last CR code has been deleted.

n < 0 : Deletes all characters preceding the CP until | n | + 1 CR codes are detected. The (| n | + 1)th CR code is not deleted.

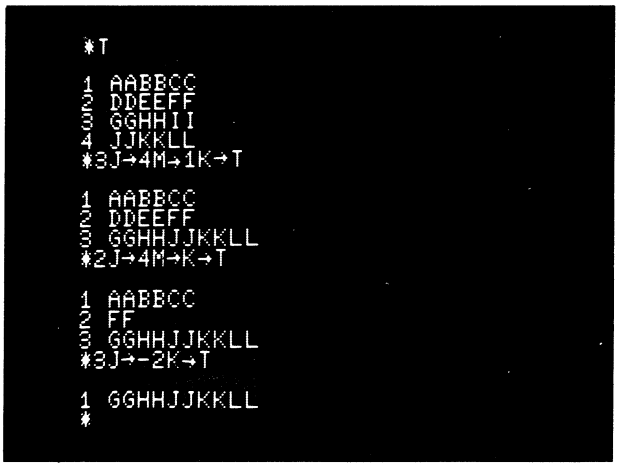
n = 0 or not specified : Deletes all characters preceding the CP until a CR code is detected. That is, it deletes the part of the line in front of the CP. The CR code detected is not deleted.

— Line numbers are automatically updated after deletion.

— The CP position is not changed.

— The photograph at right shows an example of the result of execution of the K command with the following text. (This text is presented only for the purpose of illustration; it has no meanings in assembly language.

```
1 AABBCC
2 DDEEFF
3 GGHHII
4 JJKKLL
```



D (Delete) Command

This command deletes the specified number of characters from the edit buffer, starting at the CP position.

✳nD CR	Deletes the specified number of characters from the edit buffer, starting at the CP position. A CR code is counted as a character.
✳D CR	(No operation results.)

— Key in the number of **character (n) and D.**

— Press CR to execute the command.

— Operation differs according to the value of n as follows.

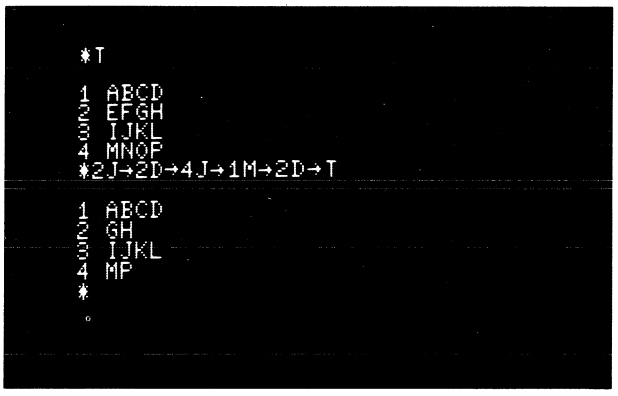
- n > 0 Deletes the n characters following the CP from the edit buffer. A CR code is counted as a character.
- n < 0 Deletes the | n | characters preceding the CP from the edit buffer. A CR code is counted as a character.
- n = 0 or not specified No operation results.

— Line numbers are automatically updated if necessary.

— The CP position is not changed.

— The photograph at right shows an example of the result of execution of the D command with the following text. (This text is presented only for the purpose of this illustration; it has no meaning in assembly language).

```
1 ABCD
2 EFGH
3 IJKL
4 MNOP
```



——SEARCH COMMAND——

S (Search) Command

- This command searches for the specified character string in the contents of the edit buffer.

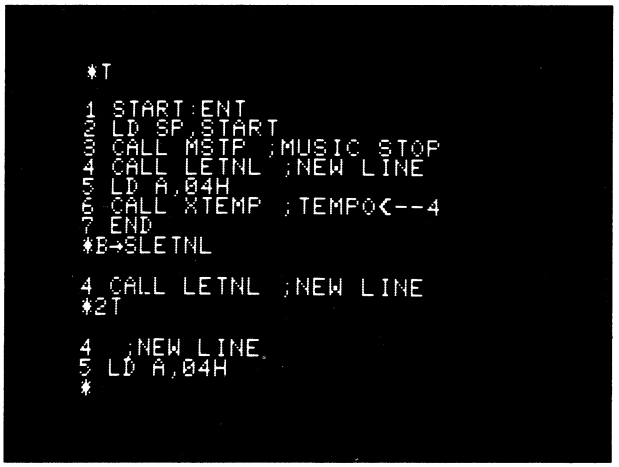
✳S string CR	Searches for the specified character **string**, starting at the current **CP** position; the **CP** is repositioned to the end of the character string when it is found.

— Key in S.
— Key in the string to be located.
— Press CR to execute the S command.

— The search starts at the current CP position and proceeds toward the end of the buffer.
— When the specified string is found, the line which includes it is displayed and the CP is positioned to the end of the character string.
— If the specified string cannot be found, the message "NOT FOUND" is displayed and the CP is repositioned to the beginning of the edit buffer.

— The photograph at right shows the result of a search for the character string "LETNL" in the following text. The line including "LETNL" is displayed following the S command. The 2T command indicates that the CP is positioned to the end of the string.

```
1 START : ENT
2 LD SP, START
3 CALL MSTP ;MUSIC STOP
4 CALL LETNL ;NEW LINE
5 LD A, 04H
6 CALL XTEMP ;TEMPO<---4
7 END
```



——SPECIAL COMMANDS——

= Command

*= CR	Displays the total number of characters (including spaces and CRs) stored in the edit buffer.

— Key in "=" (equal) while in the command wait state.

— Press CR ; the total number of characters stored in the edit buffer is displayed.

. Command

*. CR	Displays the number of the line on which the CP is located.

— Key in . (period) while in the command wait state.

— Press CR ; the line number on which the CP is located is displayed.

& Command

*& CR	Clears the contents of the edit buffer.

— Key in & (ampersand) while in the command wait state.

— Press CR ; the contents of the edit buffer are then cleared.

Command

*# CR	Changes the printer list mode.

— Key in # (sharp symbol) while in the command wait state.

— Press CR ; the printer list mode is then changed.

— The printer list mode is disabled when the text editor is started. It is enabled when the # command is executed once; executing it again disables it, and so on.

— The following shows a listing obtained by executing the T command when the printer list mode is enabled.

```
1 ;
2 ;***  EDITOR LIST SAMPLE  ***
3 ;
4 START:ENT
5 MAIN1:ENT
6 LD SP,START ;INITIAL STACK POINTER
7 CALL MSTP ;MUSIC STOP
8 LD A,5
9 CALL XTEMP ;SET TEMPO TO 5
10 CALL CLTBL ;CLEAR TABLE
11 XOR A
12 LD (?TABP),A ;INITIAL I/O #1
13 MSTP:EQU 0047H ;MUSIC STOP (MONITOR)
14 ?TABP:DEFS 1
15 END
```

! Command

*! CR	Returns control to FDOS.

— Key in ! (exclamation mark) while in the command wait state.

— Press CR ; control is then returned to FDOS.

——ERROR MESSAGE——

Error message	Explanation	Related commands
FULL BUFFER	The editor buffer is full.	R, A, PR, PA
? ? ?	n < 0 in the nT or nJ command.	T, J
LARGE	n greater than 65535 was specified.	T, J, L, M, K, D, B, Z
NOT FOUND	The string specified in the command was not found.	S, C, Q
INVALID	Other than an editor command was entered or an incorrect format was used. Ex) ＊H [CR] : There is no H command. ＊S [CR] : A string should be specified.	Any case
PAGE OPENED ?	The file to be subjected to page processing is not defined (or is not opened).	PR, PA, PW, PC

Note: Refer to the System Error Messages in the System Command manual for other system errors.
Display of the message "ALREADY OPENED" during execution of W command indicates that there are some page processing command(s) which have not been closed.

Z-80 Assembler

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

— CONTENTS —

OUTLINE OF ASSEMBLER

The assembler translates a source file written in assembly language to generate a relocatable binary file; the source file is one which has been generated and edited by the text editor, and the relocatable binary file is an intermediate file between the source file and object file. It is possible to link several relocatable files by the relocating loader.

The assembly source file is coded in assembly language. It consists of labels, mnemonic operations codes, pseudo-instructions, comments and an end statement; these are arranged according to the rules of the assembler. The source file edited by the editor is written in ASCII code. The assembler translates the source file to generate a relocatable file and outputs messages which indicate definition conditions and syntax errors. These messages are included in the assembly listing which is displayed on the CRT or printed on the printer.

The following FDOS commands activate the assembler.

- **ASM SAMPLE**

  Activates the assembler. The assembler translates source file SAMPLE.ASC and generates relocatable file SAMPLE.RB.

- **ASM SAMPLE, $LPT/L, $CRT/E**

  Activates the assembler. The assembler translates source file SAMPLE.ASC, generates relocatable file SAMPLE.RB, prints the assembly listing on the printer and displays only erroneous lines on the CRT screen.

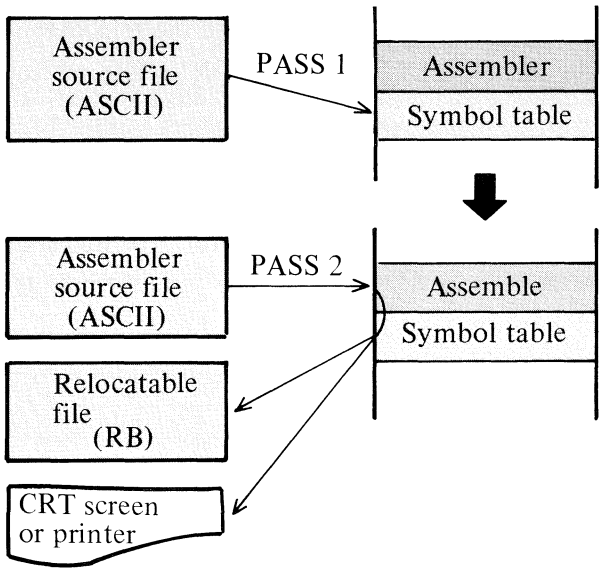
- **ASM/N SAMPLE, $SOA/L**

  Activates the assembler. The assembler translates source file SAMPLE.ASC and outputs the assembly listing to serial output port A ($SOA), but does not generate a relocatable file since global switch/N is specified.

- **ASM SAMPLE, $FD3;SAMLIST/L**

  Activates the assembler. The assembler translates source file SAMPLE.ASC, generates relocatable file SAMPLE.RB and outputs the assembly listing in the same form as that printed on the printer to SAMLIST.ASC on FD3 in ASCII code.

The assembler basically uses a 2-pass system. A pass is the process in which the assembler reads a source file from its beginning to end. The following shows operation of the assembler with the 2-pass system.



During pass 1, the assembler stores label symbols according to the assembler rules in the symbolic label table. Label symbols help the operator to read and understand the program easily.

During pass 2, the assembler generates a relocatable file with reference to the symbol table generated during pass 1, then outputs the assembly listing (on the CRT or printer).

The relocatable file and the assembly listing do not occupy space in RAM, which is only used by the symbol table. Therefore, the size of the source file to be assembled is not limited by the amount of RAM.

The following program list will help you understand the function of the assembler. This program is only for reference and has no meaning.

```
**  Z80 ASSEMBLER SP-7101 PAGE 01  **
01  0000              ;
02  0000              ; SAMPLE LIST
03  0000              ;
04  0000  3E33              LD      A, '3'
05  0002  FE43              CP      43H
06  0004  FE43              CP      'C'
07  0006  FE15              CP      "■"          ; Cursor control keys are enclosed in double quotation marks.
08  0008  22                DEFB    '"'
09  0009  27                DEFB    "'"
10  000A  43                DEFB    'C'
11  000B  12                DEFB    "■"
12  000C  16151211          DEFM    "C■■■■■■"
13  0010  1314
14  0012  7E                LD      A, (HL)
15  0013  7E                LD      A, M         ; M may be used in place of (HL).
16  0014              ;
17  0014              ;
18  0014  P         XYZ:    EQU     10
19  0014  C32100            JP      ABC + XYZ    ; Relocatable address ± EQU defined symbol value.
20  0017  C30A00    ABC:    JP      XYZ
21  001A  C31400            JP      ABC-3
22  001D  C30A00            JP      10           ; Absolute address 10
23  0020  C32A00            JP      +10          ; Relative address 2AH (20H + 10)
24  0023  2100D0            LD      HL, D000     ; Handled as a hexadecimal number.
25  0026  213930            LD      HL, 12345
26  0029  212100            LD      HL, ABC + XYZ
27  002C  3E0D              LD      A, XYZ + 3   ; EQU defined label value ± numeric data
28  002E  3EFF              LD      A, -1        ; Negative value is converted into one's complement.
29  0030  21FFFF            LD      HL, -1
30  0033  21F0FF            LD      HL, -10H
31  0036  C33500            JP      -1
32  0039                    END
**  Z80 ASSEMBLER SP-7101 PAGE 02  **
ABC     0017    XYZ     000A        ; Indicates the contents of the symbol table.
```

ASSEMBLY LANGUAGE RULES

The source program must be coded according to assembly language rules. This paragraph describes the structure of the source program and the assembly language rules.

The assembly source program consists of the following.

Z80 instruction mnemonic codes
Label symbols
Comments
Assembler directives
- **Definition statements**
- **Entry statements**
- **Skip statements**
- End statement

Comments may be used as needed by the programmer; they have no effect on execution of the program and are not included in the relocatable file.

All assembly source programs must be end with the pseudo instruction END.

Z80 instruction mnemonic codes form the body of the assembly source program. These are explained in a separate volume.

A mnemonic code consists of an op-code of up to 4 characters, separators (space, comma, etc.) and operands.

A label symbol represents an address or data symbolically. A label symbol is either placed in the label column and separated from the following instruction with a colon (:), or is placed in an operand.

The first 6 characters of a label symbol are significant and the 7th and following characters (if used) are ignored. Therefore, ABCDEFG and ABCDEFH are treated as the same label symbol.

Alphanumerics are generally used for label symbols, but any characters other than those used for separators and special symbols may be used.

Comments are written between the separator ";" and a [CR] code; these have no influence on program execution.

Pseudo instructions will be explained later in this manual. These are written in the same column as the Z80 instruction mnemonic codes.

An END statement is one of the pseudo instructions; all assembly source programs must end with this statement.

——CHARACTERS——

Characters which are used in an assembly source program are alphanumerics, special symbols and other characters. The special symbols have functional meanings. (Separators, [CR], [SPACE], etc.)

1) Alphabetic characters: ABCDEFGHIJKLMNOPQRSTUVWXYZ

These characters are used to represent symbols and instruction mnemonic codes. A ~ F are also used for representing hexadecimal values. Further, D is used to indicate decimal and H is used to indicate hexadecimal.

2) Numerics: 0 1 2 3 4 5 6 7 8 9

These are used to represent constants and symbols. Whether a constant is a hexadecimal number or a decimal number is determined according to the rule of constant.

3) Space

Spaces are treated as separators except when they are used in comments. They perform the tabulation function on the assembly listing when they are placed between op-code and operand or between operand and comment as shown below.

Example:

```
OR [SP] F0H [SP] ;A<-X0
XYZ: PUSH [SP] AF
ADD [SP] HL, BC [SP] ;BC=COUNT
```

Editor list

⬇

```
      OR     F0H      ;A<-X0
XYZ:  PUSH   AF
      ADD    HL, BC   ;BC=COUNT
             ↑        ↑
             Tab set  Tab set
```

Assembly listing

4) Colon ":"

A colon behaves as a separator when it is placed between a label symbol and an instruction. It performs the tabulation function on the assembly listing.

Example:

```
START:  LD     SP, START
MAIN:   ENT
↑       ↑
Tab set Tab set
```

An address is assigned to the label symbol even if no instruction follows. (See the paragraph on symbols).

Example:

```
ENTRY:                ← "ENTRY" is assigned the same address as "TOP0".
TOP0:   PUSH HL
```

5) Semicolon ";"

A semicolon represents the beginning of a comment. None of the characters between a semicolon and a CR code have any influence on execution of the program. The semicolon is placed at the top of a line or the beginning of a comment column.

Example:

```
;
; SAMPLE PROGRAM
;
CMMNT: ENT      ; COMMENT
```

All lines are comments.

Comment column

6) Carriage return CR

A carriage return code represents the end of a line.

7) Other special symbols: + – ' ( ) ,

All these are special symbols used in instruction statements.

8) Other symbols

Other characters are not generally used, although they may be used as symbol labels or in the comment column.

——LINE——

Each line of a source program is formed of alphanumerics and symbols, and is ended with a carriage return. Except for comments, each line includes only one of the Z80 instructions, a pseudo instruction, an end statement or an empty statement for a skip.

Components on each line are arranged according to the tab settings when it is listed. (See the assembly listing on page 7).

——LABEL SYMBOLS——

All characters other than special symbols may be used for label symbols, but generally alphanumerics are used. Each label symbol can consist of up to 6 characters; the 7th and following characters, if used, are ignored by the assembler.

Example:					
	Correct	ABC	START	BUFFER	50STEP
	Incorrect	(ABC)	,HL	IY + 3	XYZ + 3 ← Special characters are used.

The following labels are treated as the same label symbol ("COMPAR").

```
COMPARE0
COMPARE1
```

Pseudo statement EQU defines data (1 byte or 2 bytes) for a label symbol and assigns it to the label.

Example:

```
ABC:    EQU    3
CR:     EQU    0DH
VRAM0:  EQU    D000H
```

Pseudo statement ENT defines a label symbol as a global symbol. A colon (:) placed between a label symbol and a following instruction defines the label symbol as a relocatable instruction address.

Example:

```
RLDR:   ENT
RLDR0:  PUSH HL
```

When a label symbol is referenced (that is, when it is used as an operand), the assembler first searches the symbol table for the specified label symbol; if it is not found, the assembler treats it as hexadecimal data. For example, when CALL ABC is encountered, the assembler searches the symbol table for "ABC"; if it is not found, the assembler treats it as 0ABCH and calls address 0ABC.

A label symbol used as an operand must be defined in the assembly source program unit in which it is used or must be defined as a global symbol in another assembly source program unit. Otherwise, it is coverted into binary and left undefined.

A label symbol which has been defined once cannot be defined again.

Multiple label symbols may be defined as relocatable instruction addresses as follows.

Example:

```
ABCD:  ENT
EFGH:  ENT
IJK:   LD   A, B
```

Label symbols ABCD, EFGH and IJK are all defined as relocatable addresses of LD A,B. ABCD and EFGH are also defined as global symbols.

```
ABCD:
EFGH:
IJK:   LD   A, B
```

Same as the above, except that ABCD and EFGH are not global symbols.

——CONSTANTS——

There are two types of constants: decimal and hexadecimal + and – signs can be attached to these. A character string which is defined as a label symbol is treated as a label symbol even if it satisfies the requirements for a constant.

The assembler treats a constant as a decimal constant when it consists of numerics only or it consists of numerics followed by D.

Example: 23 999 +3 –62 16D 0003D

16 3

The assembler treats a constant as a hexadecimal constant when it consists of 0 ~ 9, A, B, C, D, E and/or F followed by H.

Example: 2AH CDH +01H –BH 0010H 00ADH 00H

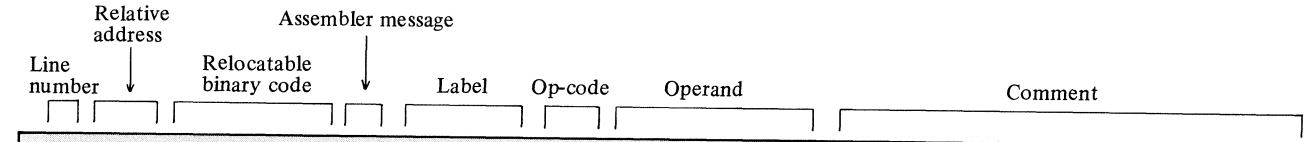
A constant used in the operand of a JP, JR, DJNZ or CALL instruction represents an absolute address when it has no sign and a location relative to the current address when it has a sign. In other cases, constants without signs and those with a + sign represent numerics, while those with a – sign are converted into two's complement.

ASSEMBLY LISTING AND ASSEMBLER MESSAGES

The assembly listing is output to the CRT screen or printer when an FDOS system command ASM is executed with $CRT/L or $LPT/L specified as an argument. Examining the assembly listing is one of the most important procedures in assembly programming since this is when a check is made for errors in the source program.

The assembler translates the specified source program and outputs the assembly listing, which includes line numbers, relative addresses, relocatable binary codes, assembler messages and the source program list (including label symbols, Z80 instruction mnemonic codes and comments). The assembly listing is pages every 50 lines.

The assembly listing format is shown below. Tabs are set at the beginnings of labels, op-codes, operands and comment columns.




```
**  Z80 ASSEMBLER SP-7101 PAGE 01  **
01  0000                    ;
02  0000                    ; ASSEMBLER LIST SAMPLE
03  0000                    ;
04  0000  P                 LETNL:  EQU  0006H
05  0000  P                 MSG:    EQU  0015H
06  0000                    ;
07  0000                    START:  ENT                ; ENTRY FROM UNIT #1
08  0000                    MAIN:   ENT                ; ENTRY FROM UNIT #2
09  0000  310000                    LD   SP, START     ; INITIAL STACK POINTER
10  0003  210000      E             LD   HL, TEMP0
11  0006  DD210000    E             LD   IX, TEMP1
12  000A  DD360000 EE   MAINO:  LD   (IX + CONST0), CONST1
13  000E  00          Q             XOA  A             ; A <--00
 :        :       :       :        :          :              :
47  005A  1A                MAIN7:  LD   A, (DE)
48  005B  B7                        OR   A
49  005C  2000        V             JR   NZ, COMP
50  005E  EB                MAIN8:  EX   DE,HL         ; EXCHANGE DE,HL

**  Z80 ASSEMBLER SP-7101 PAGE 02  **
```

This message is output at the top of each page.

A new page is started when the number of lines on the preceding page reaches 50.

Errors detected during assembly and definition conditions are indicated with assembler messages.

——DEFINITION CONDITION MESSAGES——

E (External)

This message indicates that an external symbol reference is being made; i.e., the label symbol by the operand is not defined in the assembly source program unit assembled.

The label symbol indicated must be defined as a global symbol in another assembly program unit for linkage with the current unit by the relocating loader. (See "Pseudo Instruction ENT" on page 10).

An undefined byte of data is treated as "00", 2 undefined bytes of data (or an address) are uncertain.

Example:
```
E   LD    B, CONST0
  ↑ The byte of data "CONST0" is not defined in the program unit.
E   CALL  SORT
  ↑ Address SORT is not defined in the program unit.
E E   BIT   TOP, (IY + FLAG)
↑ ↑ The byte of data "FLAG" is not defined in the program unit.
└── The byte of data "TOP" is not defined in the program unit.
```

P (Phase)

This message indicates that the label symbol is defined by an EQU statement with a constant value assigned. A label symbol indicated by this message can be referenced from an external file. In this case, however, the program unit including the EQU statement must be loaded before the other program units which are to be linked with it.

The P message is also displayed when a label symbol different from those stored in the symbol table during PASS 1 is found.

Example:
```
P   LETNL:  EQU   0006H
P   DATA1:  EQU   3
↑ Indicates that LETNL and DATA 1 are defined by EQU.
```
The P message is displayed in the relocatable binary code column rather than in the assembler message column.

——ERROR MESSAGES——

C (illegal Character error)

This message indicates that an illegal character has been used as an operand.

Example:
```
C   JP  +1000−3
```

F (Format error)

This message indicates that the instruction format is incorrect.

N (Non label error)

This message indicates that ENT or EQU has no label symbol.

Example:
```
N       EQU 0012H
```
No label symbol

L (erroneous Label error)

This message indicates that an illegal label symbol is used.

Example:
```
L       JR  XYZ
```
↑ XYZ is not defined in the current source program.

No externally defined global symbol can be used as an operand of the JR or DJNZ commands. The L message is displayed if such a label symbol is specified.

M (Multiple label error)

This message indicates that a label symbol is defined two or more times.

Example:
```
M   ABC:  LD DE, BUFFER
              ≀
M   ABC:  ENT
```
↑ Indicates that ABC is defined more than once.

O (erroneous Operand)

This message indicates that an illegal operand has been specified.

Q (Questionable mnemonic)

This message indicates that a mnemonic code is incorrect.

Example:
```
Q   CAL XYZ
```
CALL XYZ is correct.
```
Q   PSH B
```
PUSH BC is correct.

S (String error)

This message indicates that single quotation mark(s) are omitted from a DEFM statement

Example:
```
S   DEFM  GAME  OVER
```
DEFM 'GAME OVER' is correct.

V (Value over)

This message indicates that the value of the operand is out of the prescribed range.

Example:
```
V   LD    A, FF8H
V   SET   8, A
V   JR    −130
```

ASSEMBLER PSEUDO STATEMENT

Pseudo statements control assembly, but are not converted into machine language. However, in DEFB, DEFW and DEFM statements, their operands are sometimes converted into machine language.

——ENT (entry)———

This pseudo statement defines a label symbol as a global symbol. Label symbols which are referenced by two or more programs when multiple programs are linked must be defined by the entry statement.

Label symbols defined by the entry statement are included in the relocatable file so that the relocating loader can identify them. The symbolic debugger can perform symbolic addressing using these label symbols.

Label symbols which are not defined by the entry statement contribute only to assembly of the current source program unit, and are not included in the relocatable file output by the assembler. However, labels defined by the EQU statement are exceptions since they are defined as global symbols and entry definition is not necessary.

The example below shows label symbols being referenced between program units GAUSS-MAIN and GAUSS-SR.

The E message in the assembler message column indicates that a label symbol which is not defined in the current program unit is being referenced externally.

Program unit 1
"GAUSS-MAIN"

```
                    ; GAUSS-MAIN
                    ;
                    MAIN0: ENT              ← Entry definition of label symbol
Address                      :                MAIN0
 undefined                CALL CMPLX
CD0000 E                     :
 E message          CALL CMPLX+2    ← No offset can be added to a label sym-
                             :         bol which is defined externally.
                           END       ← END is always required at the end of
                                       a program unit.
```

Program unit 2
"GAUSS-SR"

```
                    ; GAUSS-SR
                    ;
Address             CMPLX: ENT          ← Entry definition of label symbol
 undefined                    :            CMPLX
C30000 E                   RET
 E message                   :
                           JP   MAIN0
                             :
                           END
```

——EQU (equate)——

This pseudo statement defines a label symbol with a numeric value (or address) assigned. The numeric value must be a decimal or hexadecimal constant. Any numeric value can be added to or subtracted from a label symbol once it is defined with a numeric value assigned; this allows a new label symbol to be defined.

The label symbol used as an address in the operand is generally treated as a relative address. However, when a specific address is assigned to the label symbol with an EQU statement, the address is not changed during assembly.

The EQU statement also defines a label symbol as a global symbol. A label defined by the EQU statement can be referenced by an external program unit. **However, program units including such statements must be loaded before other program units to be linked.**

The following example illustrates use of the EQU statement to define label symbols as monitor subroutine addresses and I/O port numbers for a specific device. The P messages indicate that the EQU statements define the label symbols as global symbols.

```
                        ** Z80 ASSEMBLER SP-7101 PAGE 01 **
01 0000                 ;
02 0000                 ;   MONITOR SUBROUTINE
03 0000                 ;
04 0000   P             PRNT:     EQU   0012H
05 0000   P             PRNTS:    EQU   000CH
06 0000   P             NL:       EQU   0009H
07 0000   P             LETNL:    EQU   0006H
08 0000   P             MSG:      EQU   0015H
09 0000   P             GETL:     EQU   0003H
10 0000   P             GETKY:    EQU   001BH
11 0000   P             BRKEY:    EQU   001EH
12 0000                           SKP   3



16 0000                 ;
17 0000                 ;   SET PORT# : PRINTER
18 0000                 ;
19 0000   P             POTFE :   EQU   FEH
20 0000   P             POTFF :   EQU   POTFE+1     ; POTFF is defined with FF (hexadecimal)
21 0000                 ;                             assigned.
22 0000   P             CON1 :    EQU   1
23 0000   P             CON2 :    EQU   2
24 0000   P             CON3 :    EQU   CON1+CON2   ; This results in assigned of 3 to CON3.
                                                      In this case, CON1 and CON2 must be
                                                      defined in advance.
```

——DEFB n (define byte)——

This statement sets constant n (1 byte) in the address of the line on which the statement is specified. A label symbol defined with a constant (1 byte) assigned may be used in place of n.

This statement (as well as DEFW and DEFM.) is used to form message data or a graphic data group for a code conversion table or other table.

The following example forms the message "ERROR" in ASCII code. Since it uses 0D as an end mark, monitor subroutine 0015H can be used to output the message.

```
13 1FF3  B7              OR     A
14 1FF4  CA0000  E       JP     Z, READY
15 1FF7  110020          LD     DE, MESG0
16 1FFA  CD1500          CALL   MSG
17 1FFD  C30000  E       JP     MAIN2
18 2000          MSG :   EQU    0015H
19 2000          ;
20 2000          ; MESSAGE GROUP
21 2000          ;
22 2000          MESG0 : ENT                  ; "ERROR"
23 2000  45              DEFB   45H
24 2001  52              DEFB   52H
25 2002  52              DEFB   52H
26 2003  4F              DEFB   4FH
27 2004  52              DEFB   52H
28 2005  0D              DEFB   0DH
```

——DEFB 'S', DEFB "S" (define byte)——

This statement sets an ASCII code corresponding to the character enclosed in single or double quotation marks in the address of the line on which the statement is specified.

Cursor control codes (CLR→←↑↓) must be enclosed in double quotation marks.

Since this statement converts characters to ASCII code, the above example can be rewritten as follows.

```
21 2000          MESG0 : ENT                  ; "ERROR"
22 2000  45              DEFB   'E'
23 2001  52              DEFB   'R'
24 2002  52              DEFB   'R'
25 2003  4F              DEFB   'O'
26 2004  52              DEFB   'R'
27 2005  0D              DEFB   0DH
28 2006  16              DEFB   "CLR"
29 2007  13              DEFB   "→"
30 2008  0D              DEFB   0DH
```

——DEFW nn′ (define word)——

This statement sets n′ in the address of the line on which the statement is specified and n in the following address; in other words, it sets two bytes of data. A label symbol may be used in place of nn′.

```
39 5FF1                  CMDT:   ENT                ; COMMAND TABLE
40 5FF1  41                      DEFB   41H
41 5FF2  0053                    DEFW   CMDA
42 5FF4  42                      DEFB   42H
43 5FF5  1E53                    DEFW   CMDB + 3
44 5FF7  53                      DEFB   53H
45 5FF8  0000    E               DEFW   CMDS
46 5FFA  0D                      DEFB   0DH
47 5FFB                  CONST0: ENT
48 5FFB  0F01                    DEFW   010FH
49 5FFD                  CONST1: ENT
50 5FFD  660D                    DEFW   0D66H
```

——DEFM 'S', DEFM "S" (define message)——

This statement sets the character string enclosed in single or double quotation marks in ASCII code in addresses starting at that of the line on which the statement is specified. The number of characters must be within the range from 1 to 64. On the assembly listing, codes for 4 characters are output on each line. Double quotation marks are used to enclose cursor control codes (CH→←↑↓).

The example on the preceding page can be written as follows with this statement.

```
21 2000                  MESG0 : ENT                ; "ERROR"
22 2000  4552524F                DEFM   'ERROR'
23 2004  52
24 2005  0D                      DEFB   0DH
25 2006  16134142                DEFM   "C→AB"
26 200A  0D                      DEFB   0DH
```

——DEFS nn' (define storage)——

This statement reserves nn' bytes of memory area starting at the address of the line on which the statement is specified.

This statement adds nn' to the reference counter contents; the contents of addresses skipped are not defined.

The following example reserves buffer areas.

```
02  4BB8                TEMPO:  ENT             ; BUFFER A
03  4BB8                        DEFS    1
04  4BB9                TEMP1 : ENT             ; BUFFER B
05  4BB9                        DEFS    2
06  4BBB                TEMP2 : ENT             ; BUFFER C
07  4BBB                        DEFS    2
08  4BBD                TEMP3 : ENT             ; BUFFER D
09  4BBD                        DEFS    128
10  4C3D                BFFR:   ENT             ; BUFFER E
11  4C3D                        DEFS    A
12  4C47                BUFFER: ENT             ; BUFFER F
13  4C47                        DEFS    2
```

The addresses are increased by amounts corresponding to the values indicated by the respective DEFS statements.

——SKP n (skip n lines)——

This statement advances the assembly listing by n lines to make the list easy to read.

```
30                      COMMON: ENT             ; NORMAL RETURN
31  3BB8    AF                  XOR   A         ; A<--00
32  3BB9    32B84B              LD    (TEMPO), A ; CLEAR CMD BUFFER
33  3BBC    110020              LD    DE, MESG0 ; "READY"
34  3BBF    C9                  RET
35  3BC0                        SKP   3
                                                        } 3 line feeds are made.
39  3BC0            ;
40  3BC0            ; ABNORMAL RETURN
41  3BC0            ;
42  3BC0            ABNRET: ENT             ; SET INVALID MODE
```

——SKP H (skip home)——

This statement advances the page during output of the assembly listing.

——END (end)——

This statement declares the end of the source program. All source programs must be ended with this statement. Assembly operation is not completed if this statement is omitted.

The assembler outputs

END?

when it reads a source file which doesn't include an END statement.

MESSAGE TABLE

Definition condition message	Meaning	Example
E (External)	Indicates that a label symbol is being referenced externally; that is, the label is not defined in the current source program unit.	E LD B, CONST0 ↑ The data byte "CONST0" is undefined. E CALL SORT ↑ The address "SORT" is undefined. E E BIT TOP, (IY + FLAG) ↑ ↑ The data byte "FLAG" is undefined. The data byte "TOP" is undefined.
P (Phase)	Defines a label symbol with a constant assigned. This message is also output when a label symbol is encountered during pass 2 which was not encountered during pass 1.	P LETNL: EQU 0006H P DATA1: EQU 3 ↑ LETNL and DATA are defined by EQU. The P message is displayed in the relocatable binary code column rather than in the assembler massage column.

Error message	Meaning	Example
C (illegal Character error)	Indicates that an illegal character is used in the operand.	C JP +1000–3
F (Format error)	Indicates that the instruction format is incorrect.	
N (Non label error)	Indicates that no label symbol is specified for ENT or EQU.	N EQU 0012H No label symbol
L (erroneous Label error)	Indicates that an illegal label symbol is used.	L JR XYZ ↑ XYZ is not defined in the current program. No externally defined global symbol can be used as the operand of a JR or DJNZ command. If such a label symbol is specified, the L message is displayed.
M (Multiple label error)	Indicates that a label symbol is defined two or more times.	M ABC : LD DE, BUFFR M ABC : ENT ↑ ABC is defined twice.
O (erroneous Operand)	Indicates that an illegal operand is specified.	
Q (Questionable mnemonic)	Indicates that the mnemonic code is incorrect.	Q CAL XYZ CALL XYZ is correct.
S (String error)	Indicates that single or double quotation mark(s) are omitted.	S DEFM GAME OVER DEFM 'GAME OVER' is correct.
V (Value over)	Indicates that the value of the operand is out of the prescribed range.	V LD A, FF8H V SET 8, A V JR –130
END?	Indicates that the END statement is missing from the source program.	

Note: Refer to the System Error Messages in the System Command manual for other system errors.

Symbolic Debugger

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

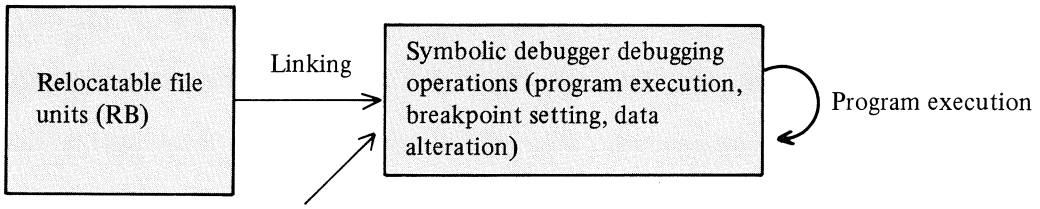
This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

— CONTENTS —

INTRODUCTION

The SHARP MZ-80K symbolic debugger links and loads one or more program units from relocatable files to form an object program in memory in an immediately executable form and runs the object program for debugging. It provides the programmer with facilities for taking a memory dump of the object program in the link area, for setting a breakpoint in the program, for displaying and altering the contents of the CPU internal registers and for starting execution of the program at a given address with the CPU internal registers set to specified values (indicative start).

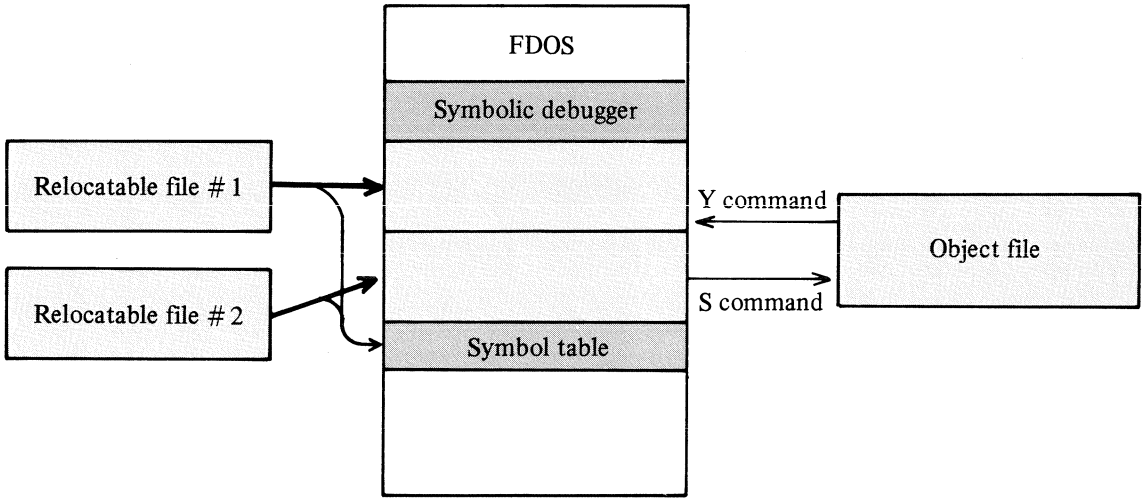


Debugging with the symbolic debugger

The debugger is said to be "symbolic" since it permits the programmer to reference addresses (e.g., breakpoints) during debugging not only in absolute hexadecimal representation but with global symbols declared as entry symbols in the source program with the ENT assembler directive. This releases the programmer from the burden of remembering relative addresses in relocatable programs and offset values specified when they are loaded.

In normal program development process, the programmer debugs each object program unit with the symbolic debugger and, if he finds errors, he reedits its source program and reassembles it. After debugging all object program units, the programmer links and loads them with the relocating loader to form the final object program.

Symbolic debugger commands are summarized in the table on page 3. Commands marked with a dagger permit symbolic operations. The debugger creates the symbol table in the same way as the relocating loader.



Symbolic debugger file processing

——STARTING THE SYMBOLIC DEBUGGER——

The symbolic debugger is started by entering one of the commands below in the FDOS command mode.

1. **DEBUG** [CR]

   The debugger is invoked and the debugger command wait state entered.

2. **DEBUG** [filename 1, ..., filename N] [CR]

   The debugger links and loads program units from relocatable files filename 1 through filename N and waits for entry of a debugger command.

3. **DEBUG/P** ABC [CR]

   The debugger loads the program unit from file ABC . RB and prints the link information shown in **Figure 1** on the printer.

4. **DEBUG/P/T** ABC [CR]

   The debugger loads the program unit from file ABC . RB and prints the link and symbol table information on the printer.

5. **DEBUG** ABC, XYZ, TBL$20 [CR]

   The debugger links and loads program units from relocatable files ABC . RB and XYZ . RB and waits for entry of a debugger command.

   It also reserves 2000 (hex) bytes (approximately 8K bytes) of space for the symbol table. Approximately 6K bytes of space are reserved when the table size is not specified.

6. **DEBUG** ABC, $1000, XYZ, DEF/O [CR]

   The debugger links and loads program units from relocatable files ABC . RB and XYZ . RB to generate an object program in object program file DEF . OBJ, then waits for entry of a debugger command. It reserves approximately 4K bytes of free space (offset of 1000 (hex)) between program units ABC and XYZ.

Note: When the debugger is invoked and the command wait state entered, all files (including those specified in the DEBUG command) are closed.

```
 LINKING ABC.RB
   TOP ASM.BIAS $7080
   END ASM.BIAS $70E7
DEBUGGER AREA 7080-B7FD
```

Fig. 1

```
 LINKING ABC.RB
   TOP ASM.BIAS $7080
   END ASM.BIAS $70E7
DEBUGGER AREA 7080-B7FD

 SYMBOL TABLE
CHR1   D 0078    CHR2   D 005D    CHR3   D 0079    CHR4   D 001D
CHR5   D 001C    CHR6   D 005C    GETKEY D 001B    MNTR   D 0000
PRINT  D 000C    STADR  D D000
```

Fig. 2

SYMBOLIC DEBUGGER COMMAND TABLE

Command type	Command name	Function
Symbol table command	**T**	Displays the contents of the symbol table; i.e., the label symbol name, its absolute address and the definition status for each table entry (Table Dump).
Debugging commands	**B** †	Displays, sets or alters a breakpoint (Breakpoint).
	&	Clears all breakpoints set (Clear Breakpoints).
	M†	Displays the contents of the specified block in the link area in hexadecimal representation or alters them (Memory Dump).
	D†	Displays the contents of the specified block in the link area in hexadecimal representation with one instruction on a line (Memory List Dump).
	W†	Writes hexadecimal data, starting at the specified address in the link area (Write).
	G†	Executes the program at the specified address (Goto).
	I	Executes the program at the address designated by PC with the register buffer data set to the CPU internal registers (Indicative Start).
	A	Displays the contents of registers A, F, B, C, D, E, H, and L in hexadecimal representation or alters them. (Accumulator)
	C	Displays the contents of complementary registers A', F', B', C', D', E', H' and L' in hexadecimal representation or alters them. (Complementary)
	P	Displays the contents of registers PC, SP, IX, IY and I in hexadecimal representation or alters them (Program Counter)
	R	Displays the contents of all registers in hexadecimal representation (Register).
	X	Transfers the specified memory block to the specified address (Transfer).
File I/O commands	**S**	Saves the object program in the link area in an output file with the specified file name.
	Y	Reads the object program from the object file with the specified file name into memory (Yank).
Special commands	#	Switches the printer list mode for listing printout.
	!	Transfers control to FDOS.

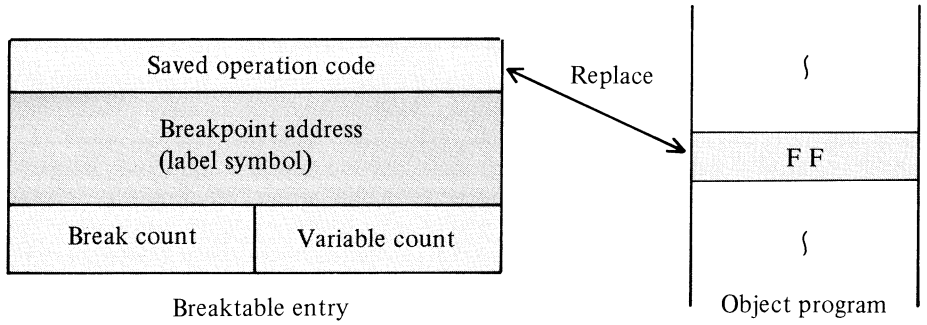
Note: Commands marked by a dagger permit symbolic operations.

BREAKPOINTS

A breakpoint is a checkpoint set up in the program at which program execution is stopped and the contents of the CPU registers are saved into the register buffer. At this point, the programmer can examine and alter the memory and register contents. He can also restart the program at this point. Thus, breakpoints facilitate program checking and debugging.

The symbolic debugger allows a maximum of nine breakpoints. When setting a breakpoint, the programmer must specify not only its address but also its count. The count specifies the number of allowable passes through the breakpoint in a looping program before a break actually occurs. The maximum allowable value of the break count is E in hexadecimal (14 in decimal).

When a breakpoint is set in a program, the debugger saves the operation code at that location (address) in the break table and replaces it with code FF. The debugger creates one breaktable entry for each breakpoint as shown below.



Hexadecimal code FF is the operation code for RST 7, which initiates a break operation. When the RST 7 instruction, which is a 1-byte CALL instruction, is executed, the contents of the program counter are pushed into the stack and the program counter is loaded with new data 0038H; that is, program control jumps to address 0038H in the monitor, from which point control is immediately passed to the debugger. The debugger searches the break table for the pertinent breakpoint. If the breakpoint is not found, the debugger displays error message "RST 7?". Thus the RST 7 instruction is used in the system for a specific purpose and cannot be used by user programs.

When the debugger finds the required breakpoint in the table, it checks the corresponding count and decrements the variable count (this count is initially set to the break count) by one. If the variable count reaches zero, the debugger performs break processing; otherwise, it continues program execution.

——T (Table Dump) Command——

The T command displays the contents of the symbol table, that is, the label symbol name, its absolute address and its definition status.

＊DT	Displays the contents of the symbol table.

— Enter a T command in response to the prompt "＊D".

— The debugger displays the label symbol name, its absolute address (in hexadecimal) and the definition status for each symbol table entry. The programmer can detect symbol definition errors by checking the definition status of the displayed label symbols.

— Messages pertaining to the symbol table definition status are identical to those issued by the relocating loader. The definition status messages are listed below, followed by examples.

Message	Definition status
U	Undefined symbol (address or data)
M	Multi-defined symbol (address or data)
X	Cross-defined symbol (address or data)
H	Half-defined symbol (data)
D	Data difinition symbol (data)

No message is attached to symbols for which an address has been defined. U, M, X and H indicate error conditions.

Message Examples

First program unit to be loaded

```
TMDLYH:   LD      HL, START
COUNT:    ENT
          DEC     HL
          LD      A, H
          CP      COUNTO
          JR      NZ, COUNT
          LD      A, L
          CP      COUNT1
          JR      NZ, COUNT
          CP      COUNT2
          JR      NZ, COUNT
          RET
PEND:     ENT
          DEFM    'TMDLYH'
          DEFB    0DH
COUNT1:   EQU     00H
COUNT0:   EQU     50H
          END
```

Second program unit to be loaded

```
TMDLYL:   LD      HL, START
LOOP1:    DEC     H
          LD      A, H
          CP      COUNT
          JR      NZ, LOOP1
          RET
PEND:     ENT
          DEFM    'TMDLYL'
          DEFB    0DH
START:    EQU     1000H
COUNT:    EQU     00H


          END
```

Third program unit to be loaded

```
INPUT:    CALL    001BH
          CALL    TMDLYL
          CALL    001BH
          LD      HL, START
          CP      0DH
          JR      Z, END
          LD      (HL), A
          INC     HL
          JR      INPUT
END:      JP      0000H
COUNT2:   EQU     12

          END
```

"START" X

START is not defined as an address in the first program but is defined as data in the second or subsequent program with the START: EQU statement.

Note:

The EQU statement should be placed at the beginning of the program unit.

"COUNT2" H

COUNT2 is not defined as data in the first program but is defined as data in the third program with the COUNT2: EQU statement.

"COUNT1" D

COUNT1 is defined as data (D indicates no error condition).

"COUNT" X

COUNT is defined as an address in the first program and simultaneously as data in the second program.

"PEND" M

PEND is defined as an address in the first program and simultaneously as an address in the second program (duplicated definition).

"TMDLYL" U

TMDLYL is neither defined as an address nor declared with the ENT assembler directive in any other external program unit.

—— B (Breakpoint) Command ——

The B command sets or changes a breakpoint. A breakpoint occurs after instructions immediately preceding the breakpoint are executed the number of times specified in the break counter. When a breakpoint is taken, program execution is interrupted and control is passed to the debugger. The debugger saves the contents of the CPU registers into the register buffer and waits for a debugger command. The programmer can specify the breakpoint with either an absolute hexadecimal address or a label symbol (the label symbol can be given a displacement of from–65535 to 65535 in decimal).

***DB**	Sets a breakpoint.
ADDR COUNT	
1 **7530**␣**2**	The breakpoint is address **7530** and the break count is 2.
2 **SORT 3**␣**1**	The breakpoint is the address represented by label symbol "**SORT3**" and the break count is 1.
3 **SORT 3+5L**␣**1**	The breakpoint is the address of the instruction 5 lines away from the address represented by label symbol "**SORT3**" and the break count is 1.
4 **MAIN0–9**␣**2**	The breakpoint is the address of the instruction 9 bytes before the address represented by label symbol "**MAIN0**" and the break count is 2.
5 ■	(The breakpoint and the break count must be separated by at least one blank (denoted by ␣).)

— Enter the B command in response to the prompt "*D".

— The debugger carries out a new line operation and displays "ADDR COUNT." It then performs a new line operation and displays the breakpoint number followed by a space and the cursor to prompt the programmer to enter a breakpoint address and a break count.

The programmer may specify a breakpoint address with a 4-digit hexadecimal number or a global symbol (see the examples above). In either case, enter an address followed by a space and a break count. The break count specifies the number of allowable passes through the breakpoint before a break actually occurs. **The programmer can specify a hexadecimal value from 1 to E.**

When a break count is entered, the debugger performs a new line operation and displays the next breakpoint number to prompt for the next breakpoint address.

— When a label symbol is entered as a breakpoint address, the debugger displays message "???" and waits for a new command if the pertinent symbol is not defined or if the symbol is a data difining symbol.

— No breakpoint can be specified for the DJNZ instruction. When a breakpoint is specified for the DJNZ instruction, the debugger displays message "DJNZ?" and waits for entry of a new command.

— No breakpoint can be specified for the CALL instruction either. Breakpoints cannot be specified for any instructions which push the program counter contents into the stack. The debugger will display the message "CALL?" if such an attempt is made.

To check a CALL instruction, set a breakpoint at the beginning of the called routine.

— To clear a previously set breakpoint, enter that breakpoint address with a break count of 0 (use the & command to clear all breakpoints).

The debugger displays message "???" and waits for a command when an attempt is made to clear an undefined breakpoint.

— **The programmer can specify a maximum of nine breakpoints.** When the programmer specifies nine breakpoints, the debugger displays "X" on the next line instead of the next breakpoint number. This requests the programmer to clear a breakpoint or change a break count, not to set a new breakpoint. If the programmer attempts to set a new breakpoint, the debugger will not accept it and prompts for a new command with message "OVER".

— When a B command is entered after breakpoints are set, the debugger displays them; in this case, the hexadecimal address is displayed first, followed by the break count format.

— The programmer can use the cursor key while setting breakpoints. When the CR key is pressed, the debugger is returned to the command wait state.

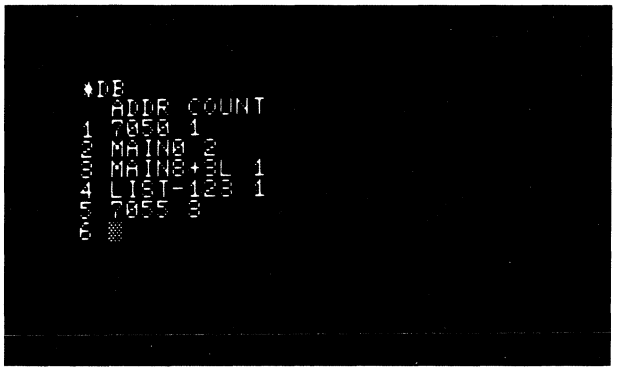
—— & (Clear B.P) Command ——

*D&	Clears all the breakpoints which have been set.

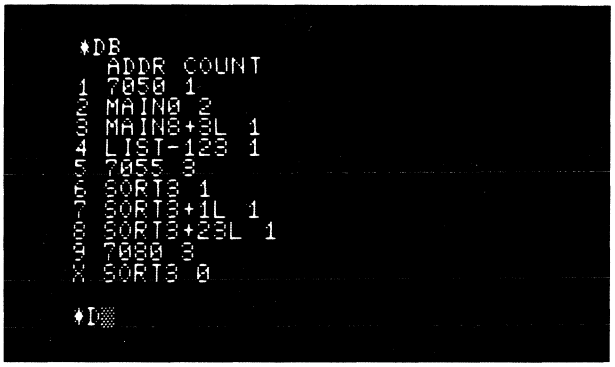
— Enter the & command in response to the prompt "*D".

— The debugger clears all breakpoints set and waits for entry of a new command.

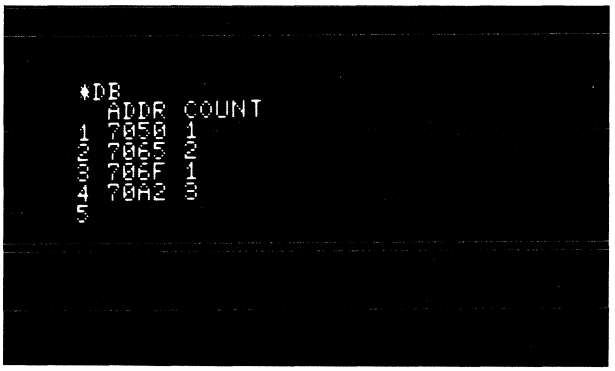
— The photo at right shows an example of setting breakpoints. The breakpoints are set with a 4-digit hexadecimal number (absolute address), a global label symbol, a label symbol plus a line specification and a label symbol plus a byte displacement.



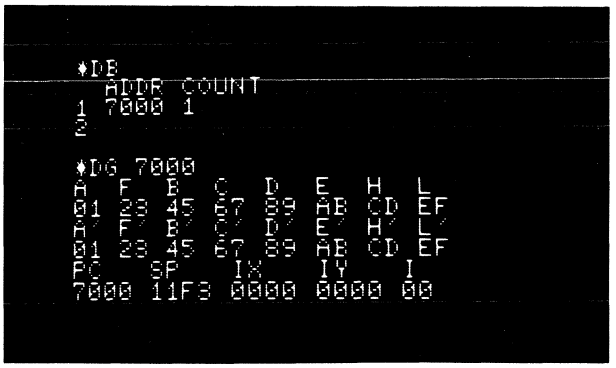
— The photo at right shows that breakpoint "SORT3" has been cleared on the line identified by "X".



— The photo at right shows an example of displaying previously set breakpoints with a B command. Breakpoints are displayed with hexadecimal absolute addresses shown first, followed by the break counts.



— The photo at right shows that a break occurred immediately when the program was executed from address 7000 with a G command with a breakpoint at 7000 and a count of 1. As soon as a breakpoint was taken, an R command was executed to display the status of the CPU registers.



—— M (Memory Dump) Command ——

The M command displays the contents of the specified memory block in hexadecimal representation. The memory block may be specified with either absolute hexadecimal addresses or label symbols. The M command permits the programmer to alter data with the cursor.

***DM 7800 _7850 [CR]**	Displays the contents of the memory block from **7800** to **7850.**
***DM MAIN7 _ MAIN9 [CR]**	Displays the contents of the memory block from the address identified by **"MAIN7"** to the address identified by **"MAIN9".**
***DM STEP0–9 _ STEP3 + 15L [CR]**	Displays the contents of the memory block from the address **9** bytes before the address identified by label symbol **"STEP0"** to the address of the instruction **15** lines away from label symbol **"STEP3".**

— Enter the M command in response to the prompt "*D".

— The debugger displays the cursor with a space between the cursor and the letter M and waits for the programmer to enter the starting and ending addresses of the memory block to be dumped. The programmer may specify the starting and ending addresses of the memory block with either 4-digit hexadecimal numbers or global symbols.

— **The starting address must be smaller than or equal to the ending address.** Otherwise, the debugger will display the message "?".

— When a memory block in the link area is specified, the debugger displays a dump of memory contents on the screen with 8 bytes on a line.

— If the printer is placed in the enable mode, the debugger prints the memory dump on the printer with 16 bytes on a line.

— The cursor appears on the screen when the memory block dump is completed. The programmer can then alter byte data in the memory dump by moving the cursor to the desired byte position on the screen, entering the new byte data and pressing [CR]. The byte data under the cursor is overwritten with the new data. The debugger displays the message "ERROR" if the data entered does not match the byte format.

— When [CR] is pressed with the cursor on a memory dump line, the data on that line is reentered into memory. The debugger is returned to the command mode, however, when [CR] is pressed with the cursor at a line containing no data.

— Press the [SPACE] key to suspend display of the memory dump. To resume display, press the [SPACE] key again.

— Press the [SHIFT] and [BREAK] keys simultaneously to force the debugger into the command mode.

—— D (Memory List Dump) Command ——

The D command displays the contents of the specified memory block in hexadecimal representation with one instruction on a line. The memory block may be specified with either absolute hexadecimal addresses or label symbols. The programmer cannot alter memory contents through cursor manipulation.

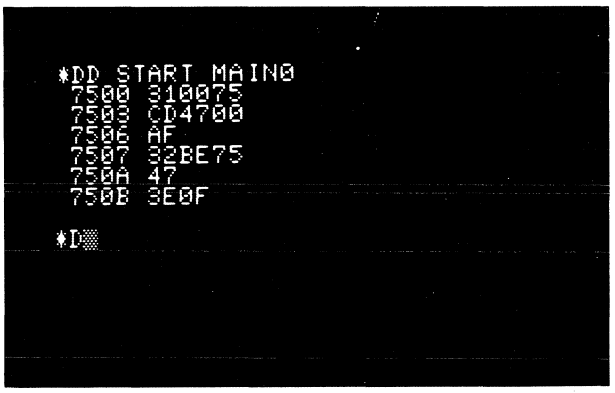
***DD 7800–7850** CR	Displays the contents of the memory block from addresses 7800 to 7850 with one instruction on a line.
***DD START–MAINO** CR	Displays the contents of the memory block from the address identified by "**START**" to the address identified by "**MAIN0**" with one instruction on a line.
***DD 7500–START + 12L** CR	Displays the contents of the memory block from address 7500 to the address of the instruction 12 lines away from the label symbol "**START**" with one instruction on a line.

— Enter the D command in response to the prompt "*D".

— The debugger displays the cursor with a space between it and the letter D, then waits for the programmer to enter the starting and ending addresses of the memory block to be dumped. The programmer may specify the starting and ending addresses of the memory block either with 4-digit hexadecimal numbers or global symbols. As with the M command, **the starting address must be smaller than or equal to the ending address.**

— Press the CR key after specifying the required memory block; the debugger then displays an address and machine language code on each line.

Consider the source program shown below, which contains the label symbols "START" and "MAIN0". Assume that the corresponding object code is loaded in memory starting at address 7500. When a D command is entered, the debugger displays a dump listing on the screen as shown in the photo at right.




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

— It must be noted that the memory block starting address specified in the D command must contain an operation code. If the starting address contains a data byte, subsequent lines dumped will display meaningless instructions which read that data byte as an operation code. The same note applies to the data areas (defined by DEFB and DEFW, etc.) in the memory block.

— Display of the memory dump listing can be suspended and resumed with the [SPACE] key.

— The D command does not allow memory alteration; after the memory dump is completed, the debugger is returned to the command wait state.

— Press the [SHIFT] and [BREAK] keys simultaneously to terminate this command in the middle of a dump.

——W (Data Write) Command——

The W command writes hexadecimal data, starting at the specified memory address. The memory address may be either an absolute hexadecimal address or a label symbol.

***DW 8000 [CR]**	**Writes machine language data, starting at address 8000.**
***DW DATA 1 [CR]**	**Writes machine language data, starting at the address identified by label symbol "DATA 1".**

— Enter the W command in response to the prompt "*D".

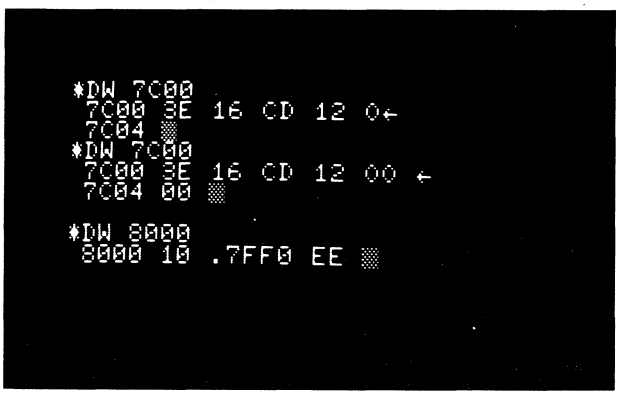
— The debugger displays the cursor with a space between it and the letter W, then waits for the programmer to enter the starting address of the memory area to be written.
The programmer may specify the memory block starting address with a 4-digit hexadecimal number or a global symbol.

— The memory area to be written must be inside the link area.

```
*DW 1111
  1111
  ? ? ?
```
} Address 1111 is not in the link area.

— When the programmer presses the [CR] key after specifying an address, the debugger displays that address on the next line to prompt the programmer to enter 2-digit hexadecimal data.
The debugger enters a space each time 2-digit data is entered and performs a new line operation and displays a new address each time eight bytes of data are entered.

— To correct the data just entered, press the ← key to return the cursor to the byte of data just entered and correct it. The photo on the right shows an example.
As the photo shows, when the ← key is pressed, the cursor is placed on the next line and the address of the byte of data to which the cursor is moved is displayed.



— To specify a displacement for a JR, DJNZ or other Z80 relative jump instruction, enter a period; the debugger waits for the programmer to enter an absolute address (no label is allowed) with a 4-digit hexadecimal number as the destination of the jump. When the programmer enters a 4-digit hexadecimal address, the debugger computes the displacement and stores the 1-byte result in the current byte position.

The seventh and eighth lines in the photo on page 12 show an example of specifying a displacement.

— After the necessary data has been written, press [CR]; the debugger then returns to the command wait state.

—— G (Goto) Command ——

The G command transfers program control to the specified address. This command is also used to restart the program following a breakpoint.

***DG 7700 [CR]**	Executes the program at address **7700.**
***DG START [CR]**	Executes the program at the address identified by label symbol **"START".**
***DG [CR]**	Restarts the program at the breakpoint. The restart address and **CPU** register data are sotred in the register buffer.

— Enter a G command in response to the prompt "*D".

— The debugger then waits for entry of an execution address. The programmer can specify the execution address with either a 4-digit hexadecimal number or a global label symbol defined with the ENT assembler directive.

When using a label symbol, the programmer can specify the execution address on a line or byte basis.

***DG MAIN0**	Executes the program at address **"MAIN0".**
***DG MAIN0 + 3L**	Executes the program at the address 3 lines after **"MAIN0"**
***DG MAIN0 – 12**	Executes the program at the address 12 bytes before the address identified by **"MAIN0".**

— To restart the program at a breakpoint, enter a G command and press [CR]. If this operation is initiated when no breakpoint is taken, the debugger returns to the command wait state without executting the program.

The contents of the CPU registers to be restored when the program is restarted are displayed with the R command. The value in the program counter (PC) is used as the restart address. Since the PC value can be changed with the P command, it is possible to restart the program at an address other than the breakpoint.

— Press the [SHIFT] and [BREAK] keys simultaneously to terminate entry of a G command.

—— I (Indicative Start) Command ——

The I command executes the program with the CPU registers loaded with the register buffer contents. The execution address is designated by the program counter. The contents of the CPU registers can be specified by the programmer through use of the A, C and P commands.

```
*DI
 A   F   B   C   D   E   H   L
 01  23  45  67  89  AB  CD  EF
 A'  F'  B'  C'  D'  E'  H'  L'
 01  23  45  67  89  AB  CD  EF
 PC      SP      IX      IY      I
 78AB    1FEA    5F70    5F50    00
 START OK?■
```

Executes the program at the address designated by the program counter with the data shown on the screen loaded in the CPU registers.

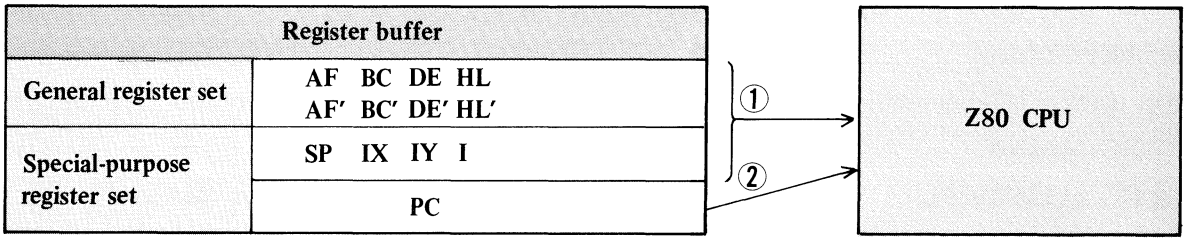
— Enter the I command in response to the prompt "*D".

— The debugger displays the 2- and/or 4-digit hexadecimal values to be loaded into the CPU registers. These values are stored in the register buffer. They can be displayed with the R command.

— The debugger then displays message "START OK?". To start the program in this environment, press [CR]. The debugger then executes the program, starting at the address designated in the program counter.

To change register values or terminate the I command, press the [SHIFT] and [BREAK] keys simultaneously; the debugger then returns to the command wait state.

— The figure below shows how the CPU registers are set with the I command.



The CPU general registers and special-purpose registers SP, IX, IY and I are loaded first; the program counter is then loaded with the execution address and the program is executed.

— The photo at right shows how the debugger responds to the I command and executes the program (at address 7500 in this example).

——A (Accumulator) Command——

The contents of the Z80 CPU registers are saved in the register buffer when a breakpoint is taken; the contents of the primary registers saved can be displayed with the A command. The buffer contents can also be altered using cursor manipulation.

```
*DA
 A  F  B  C  D  E  H  L
 01 23 45 67 89 AB CD EF
 ■
```

Displays the contents of primary register pairs AF, BC, DE and HL.

— Enter the A command in response to the prompt "*D".

— The debugger displays the contents of accumulator A, flag register F and general register pairs BC, DE and HL with 2-digit hexadecimal numbers. These values represent the contents of the primary CPU registers set up when a break occurs at a breakpoint. They are stored in the register buffer for use in subsequent restart operations (see the G command description) at the breakpoint.

— The debugger displays the cursor on the line following the one last displayed. If necessary, the programmer can alter the register contents. To change a register value, place the cursor on the desired register value, overwrite it with a new valûe, and press [CR] (the cursor will move the the beginning of the next line).

The register values displayed or altered with the A command are those values which will be restored to the CPU internal registers on a restart at a breakpoint or on an indicative start with the I command.

— Press [CR] with the cursor on the new line; the debugger then returns to the command wait state.

——C (Complementary) Command——

The C command displays the contents of the complementary general-purpose registers set up on the last break. The programmer can alter their contents through cursor manipulation.

```
*DC
 A' F' B' C' D' E' H' L'
 01 23 45 67 89 AB CD EF
 ■
```

Displays the contents of complementary register pairs AF', BC', DE' and HL'.

— Enter the C command in response to the prompt "*D".

— The debugger displays the contents of accumulator A', flag register F' and general-purpose register pairs BC', DE' and HL' with 2-digit hexadecimal numbers. The contents of the registers and the meanings of the register contents and data altered through cursor manipulation are the same as with the A command. They are used for restart at a breakpoint or with the I command.

— Press the [CR] key with the cursor on the new line; the debugger then returns to the command wait state.

——P (Program Counter) Command——

The P command displays the contents of the special-purpose registers set up on the last break. The programmer can alter their contents through cursor mainipulation.

```
*DP
 PC    SP    IX    IY   I
 78AB  1FEA  5F70  5F50 00
 ■
```

Displays the contents of special-purpose registers **PC, SP, IX, IY and I.**

— Enter the P command in response to the prompt "*D".

— The debugger displays the contents of special-purpose registers PC, SP, IX, IY and I with 2- and/or 4-digit hexadecimal numbers. The meanings of the register contents and the data altered through cursor manipulation are the same as with the A and C commands.

The register values displayed or altered through cursor manipulation are restored into the pertinent registers upon restart at a breakpoint or upon indicative start with the I command. The program does not have to restart at the breakpoint; **the programmer can specify another restart address by altering the PC value.**

— Press [CR] with the cursor on the new line; the debugger then returns to the command wait state.

——R (Register) Command——

The R command displays the contents of all CPU internal registers set up on the last break or altered with the A, C or P commands. The programmer cannot alter their contents.

```
*DR
 A   F   B   C   D   E   H   L
 01  23  45  67  89  AB  CD  EF
 A'  F'  B'  C'  D'  E'  H'  L'
 01  23  45  67  89  AB  CD  EF
 PC     SP      IX     IY    I
 78AB   1FEA    5F70   5F50  00
```

Displays the contents of all **CPU** registers.

— Enter the R command in response to the prompt "*D".

— The debugger displays the contents of all CPU registers with 2- and/or 4-digit hexadecimal numbers. The cursor does not appear in the screen, so the programmer cannot alter their values.

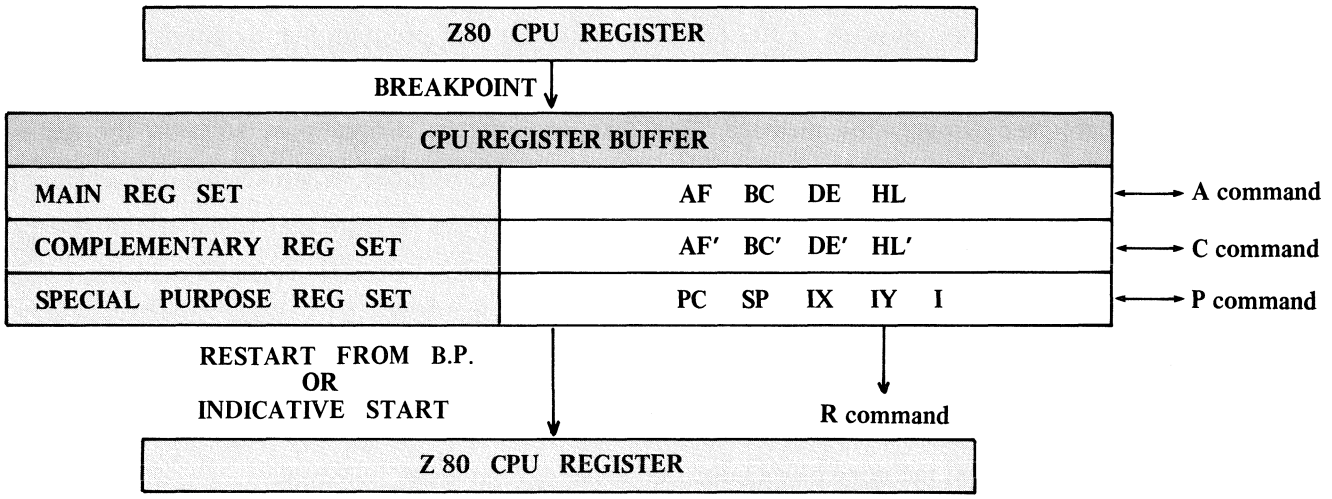
The same data is automatically displayed when a break occurs or when an indicative start is initiated with the I command.

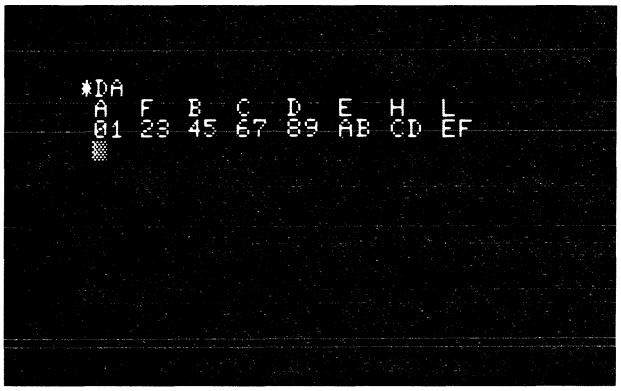
— The debugger enters the command wait state after displaying the all register contents.

Using Register Commands A, C, P and R

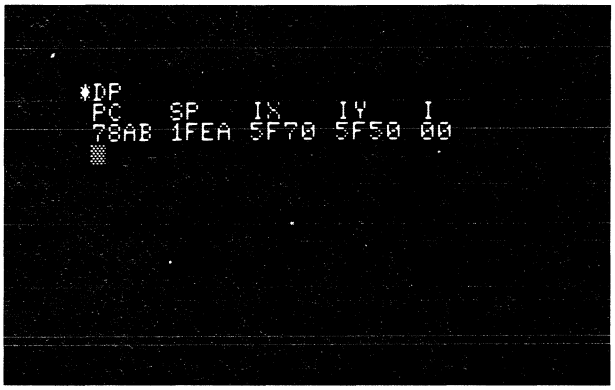
Values displayed with register commands (A, C, P and R) are the actual contents of the register buffer in the debugger. The register buffer in the debugger contains values loaded when breaks occur or when changes are made through cursor manipulation with the A, C or P command. The values are restored the CPU registers when a restart is made from a breakpoint or when an indicative start is made.

The figure below shows the relationship between the CPU registers and the register commands; the photos show examples of use of the register commands.





A command



P command

C command

R command

——X (Data Transfer) Command——

The X command transfers the contents of the specified memory block to the specified memory area.

***DX** **FROM? 7500 TO? 811F TOP? 9000**	Transfers the contents of the memory block from addresses 7500 to 811F to the memory area starting at address 9000.

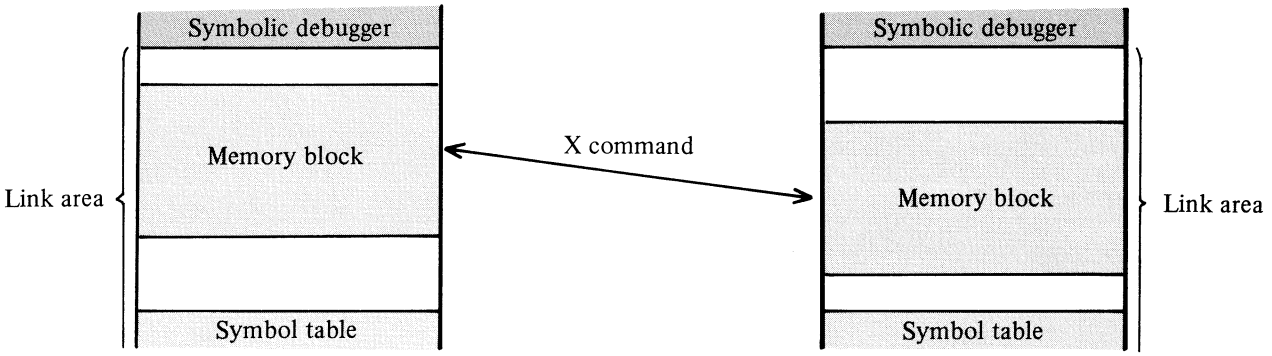
— Enter the X command in response to the prompt "*D".

— The debugger displays the message "FROM?" and waits for the programmer to enter the starting address of the source memory block with a 4-digit hexadecimal number. When the starting address is entered, the debugger displays the message "TO?" to prompt the programmer to enter the ending address of the source memory block with a 4-digit hexadecimal number. When the ending address is entered, the debugger displays the message "TOP?" to prompt the programmer to enter the starting address of the destination memory area with a 4-digit hexadecimal number (symbolic addresses are disallowed).

— When the last address is entered, the debugger starts transferring the memory block. After completing the transfer, it returns to the command wait state.

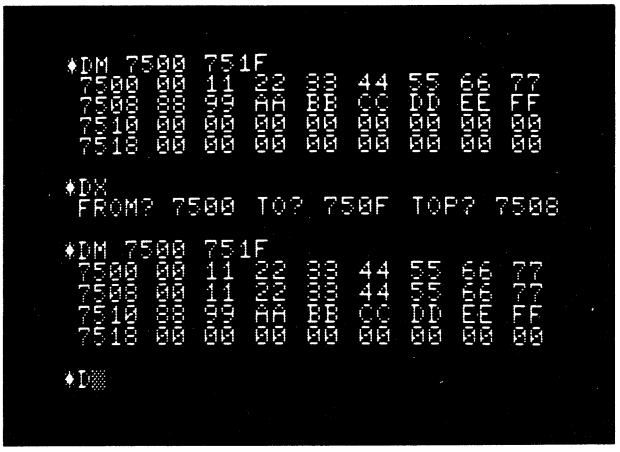
— The source and destination memory blocks must be located within the link area.

— Data transfer is accomplished successfully even if the source and destination memory blocks overlap as shown below. The memory block shown in the figure at left may be transferred to the memory block shown in the figure at right and vice versa.



— The photo at right shows how the debugger transfers the memory block starting at address 7500 and ending at address 750F to the memory area starting at address 7508.

Compare the memory contents displayed with the two M commands.



——S (Save) Command——

The S command saves a specified block of the object program in the symbolic debugger link area into a named output file in immediately executable form. The contents of this file can be restored to the link area with the Y command.

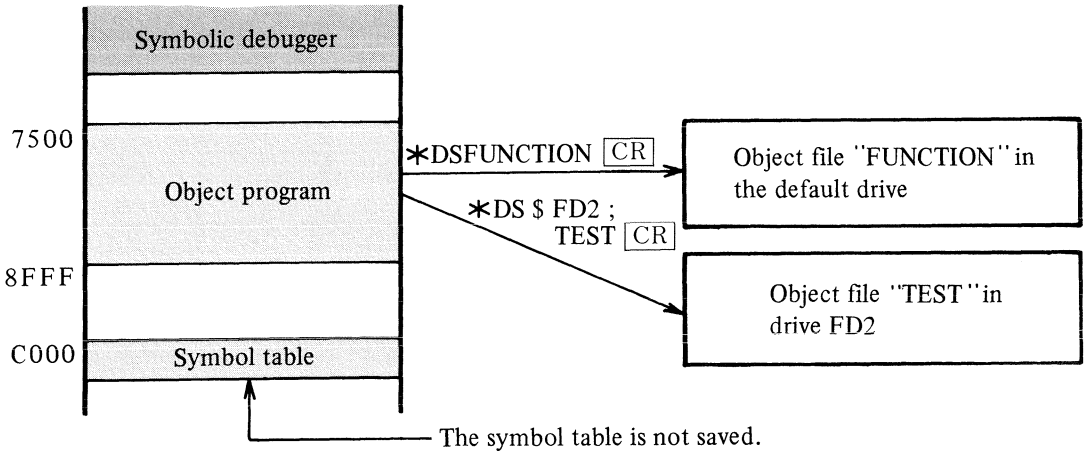
***DSfilename** [CR] **TBE _ 7500 _ 8FFF _ 7500** [CR]	Saves the immediately executable object program from addresses 7500 to 8FFF in the link area to an object file with a file name of filename. OBJ.

— Enter the S command followed by a file name in response to the prompt "*D".

— Press [CR] after entering a file name. The debugger displays TBE (Top-Bottom-Execute) message after verifying that the specified file does not exist on the specified diskette.

— Enter the starting and ending addresses of the block to be saved and the execution address with 4-digit hexadecimal numbers or symbolic label names. When the execution address is omitted, the debugger assumes the block starting address as the execution address.

— The figure below shows how the object program block from addresses 7500 to 8FFF is saved to an output file with the file name "FUNCTION".



——Y (Yank) Command——

The Y command reads the object file identified by filename into the link area.

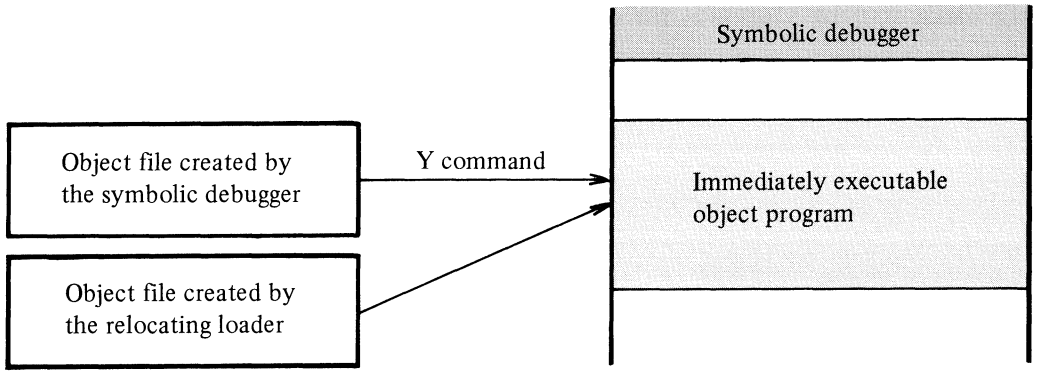
***DY filename** CR	Reads the object file named filename into the link area under loading conditions established when the file was saved.

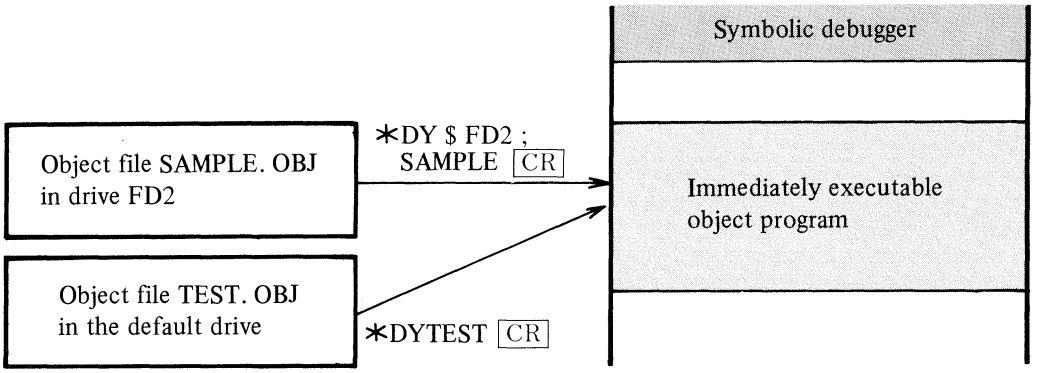
— Enter the Y command followed by a file name in response to the prompt "*D".

— Press CR after entering the file name. The debugger then searches for the file named filename. OBJ and reads it.

— The program in the filename. OBJ file is loaded into the link area block between the starting and ending addresses specified when the file was saved with the S command.

Note: Files opened before the Y command is issued are all killed.





—— # Command ——

*D #	Switches the list mode for printout on the printer.

— Enter the # command in response to the prompt "*D".

— The debugger then switches the list mode. When the debugger is invoked, the printer list mode is set to the disable mode. The mode alternates between enable and disable each time a # command is entered. In the enable mode, all output is directed to both the screen and the printer (except with the M command).

—— ! Command ——

*D !	Returns control to **FDOS**.

— Enter the ! command in response to the prompt "*D".

— Control is then transferred to FDOS.

ERROR MESSAGES

Error message	Description	Related commands
? ? ?	○ The command operand fields does not match the 4-digit hexadecimal format. ○ A symbolic label is missing. ○ A data defining symbol is used as a label.	M, D, W, B, G
ERROR	○ An invalid number of digits was entered when altering register or memory contents, or a key other than 0 through 9 or A through F was pressed.	A, C, P, M
DJNZ?	○ A breakpoint was set for a DJNZ instruction.	B
CALL?	○ A breakpoint was set for a CALL instrctuion.	B
RST 7?	○ A breakpoint was set for a RST 7 instruction.	B
OVER	○ An attempt was made to set more than 9 breakpoints.	B
?	○ An attemp was made to access outside the link area. ○ The starting address is greater than the ending address. ○ An attempt was made to clear an undefined breakpoint. ○ The breakpoint counter was set to F (the maximum permissible value is E in hexadecimal).	M, D, W, B, G, X M, D B B

Refer to the "System Command" manual for other system error messages.

Relocating Loader

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

— CONTENTS —

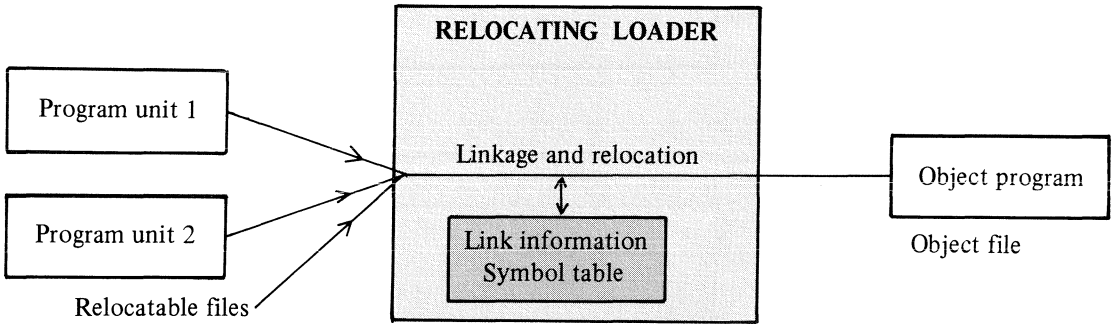
INTRODUCTION

The relocating loader for the SHARP MZ-80K inputs relocatable files output by the assembler and outputs object programs (object files).

Relocatable files are not programs which are directly executable by the CPU, but are files which contain information used to keep programs relocatable. They also contain global symbols in ASCII code which are declared to link two or more program units.

The relocating loader fetches relocation information and loads object programs into the link area in main memory while adding the programmer-specified loading address to the relocatable addresses. When two or more relocatable program units are loaded, units are appended to the first program unit (file), if the loading address is specified for the first unit.
The link area is allocated by the relocating loader; it cannot be specified by the programmer. It is used temporarily by the relocating loader and it does not necessarily conform with the address format of the object program. The linkage operation itself is described in detail in Section 2.3, "Relocating Loader" of the "System Command" manual. However, the programmer does not need to be aware of details of the linkage operation details.

When outputting the object program (object file), it is necessary to specify the loading address and the execution address.

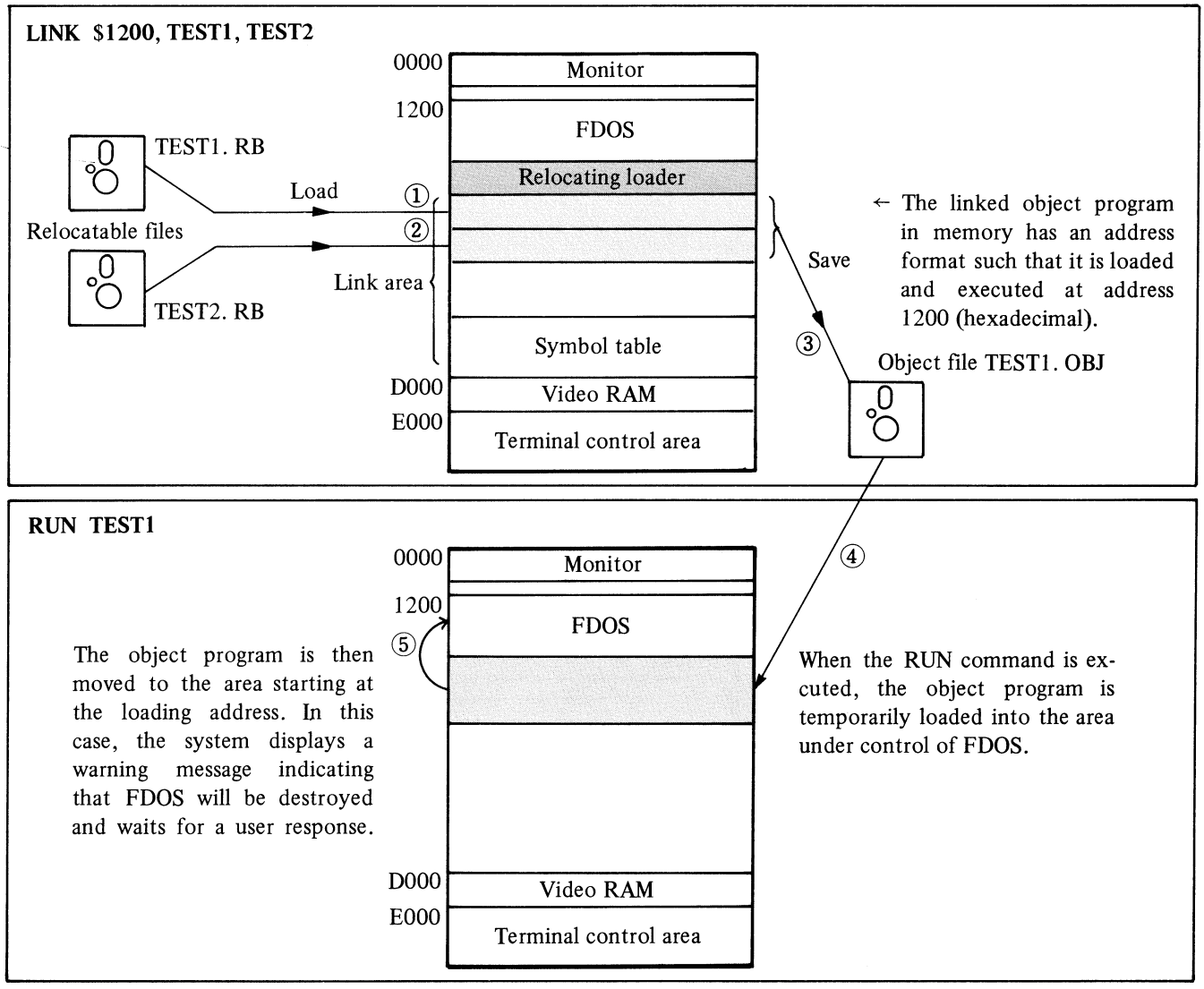


——LOADING ADDRESS——

The loading address specifies the address at which the object program is to be loaded. When this address is not specified, FDOS assumes the starting address which can be managed by FDOS as loading address.

LINK TEST1, TEST2	Links TEST1 and TEST2 and assigns the loading address to the beginning of the area managed by FDOS.
LINK $1200, TEST1, TEST2	Links TEST1 and TEST2 and assigns the loading address to 1200H.

The figure below shows the flow of files from the time they are linked by the relocating loader until they are executed with the RUN command. Numbers ① through ⑤ in the figure denote the processing sequence.



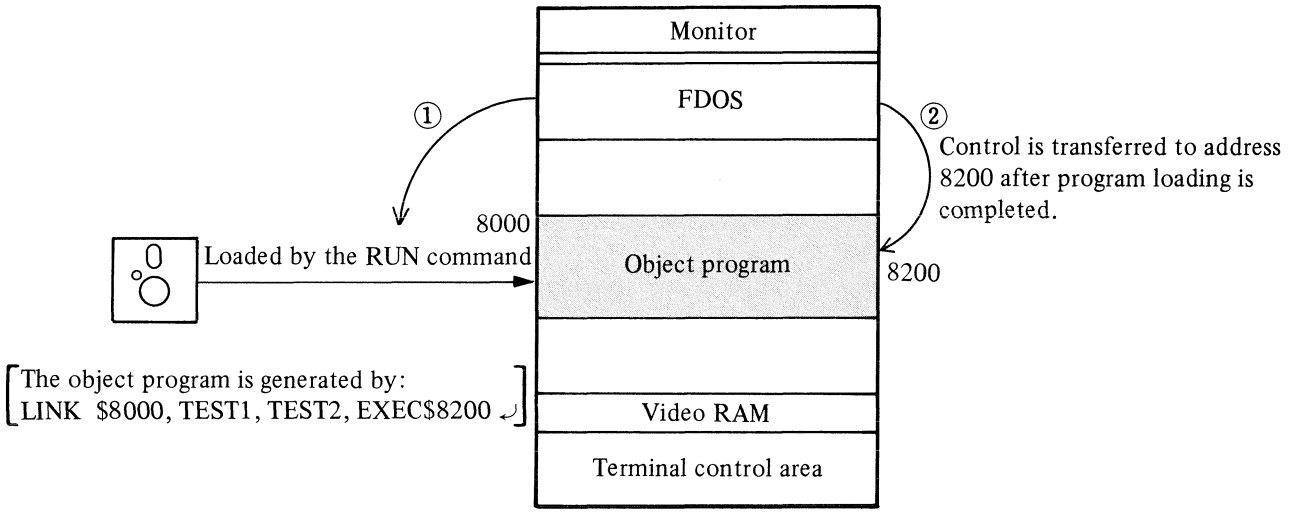
——RELATIONSHIP BETWEEN THE EXECUTION ADDRESS AND LOADING ADDRESS——

The programmer may specify the execution address as well as the loading address when outputting an object file through the relocating loader.

```
LINK $8000, TEST1, TEST2, EXEC$8200
```

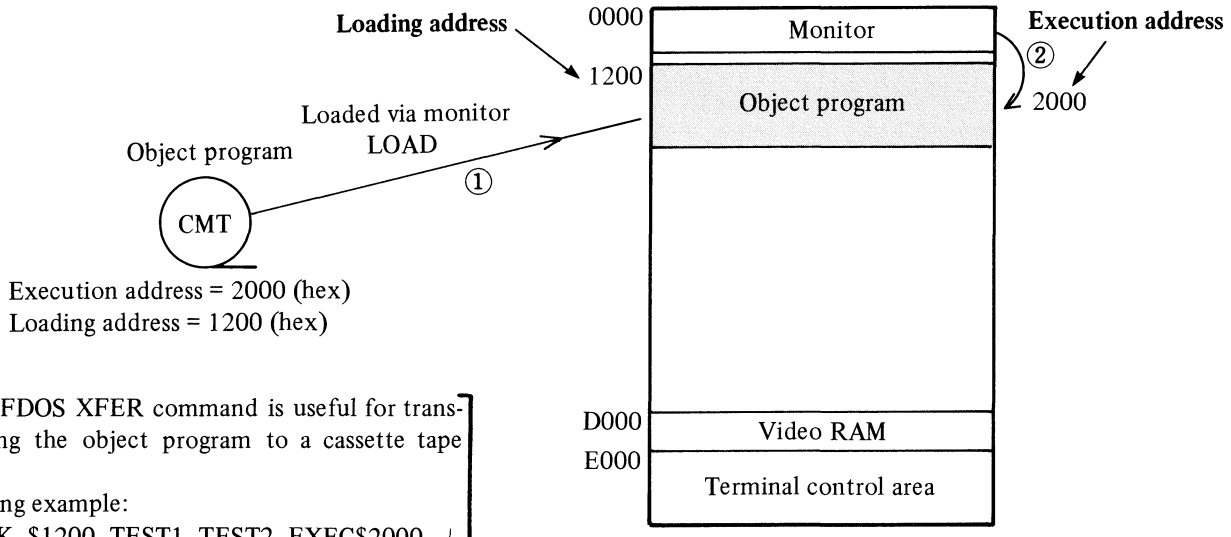
The above command links and loads relocatable program unit files TEST1 and TEST2 into memory, specifying a loading address of 8000 (hex) and an execution address of 8200 (hex).

Examples of linkage and loading are given below (numbers in circles in the figures denote the processing steps). The first example uses a simple RUN command.



Memory after loading with the FDOS RUN command

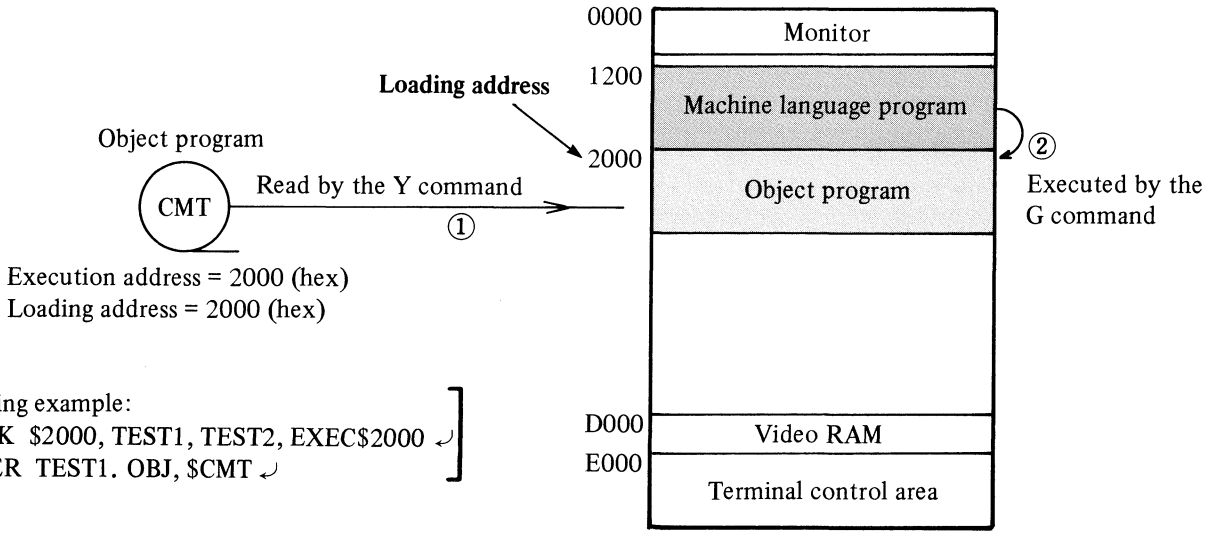
When the monitor is used to load the object program, its starting address in memory is designated by the loading address. The program counter is set to the address designated by the execution address after the object program is loaded. The figure below shows how an object program with a loading address of 1200 and an execution address of 2000 is loaded and how control is transferred.



The FDOS XFER command is useful for transferring the object program to a cassette tape file.
Coding example:
LINK $1200, TEST1, TEST2, EXEC$2000 ↵
XFER TEST1. OBJ, $CMT ↵

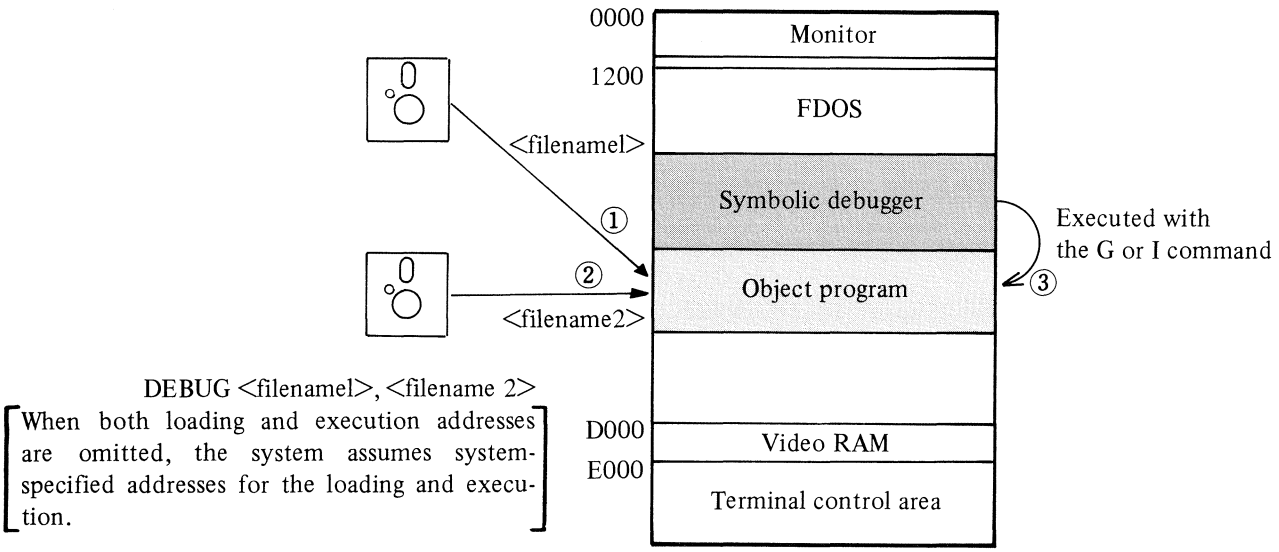
Memory after loading with the monitor program

When the object program is loaded by the symbolic debugger or the machine language system command instead of the monitor, the execution address is ignored and control is retained by the system program. To execute the program, it is necessary to transfer program control to the required execution address using the system program G command.



Coding example:
LINK $2000, TEST1, TEST2, EXEC$2000 ↵
XFER TEST1. OBJ, $CMT ↵

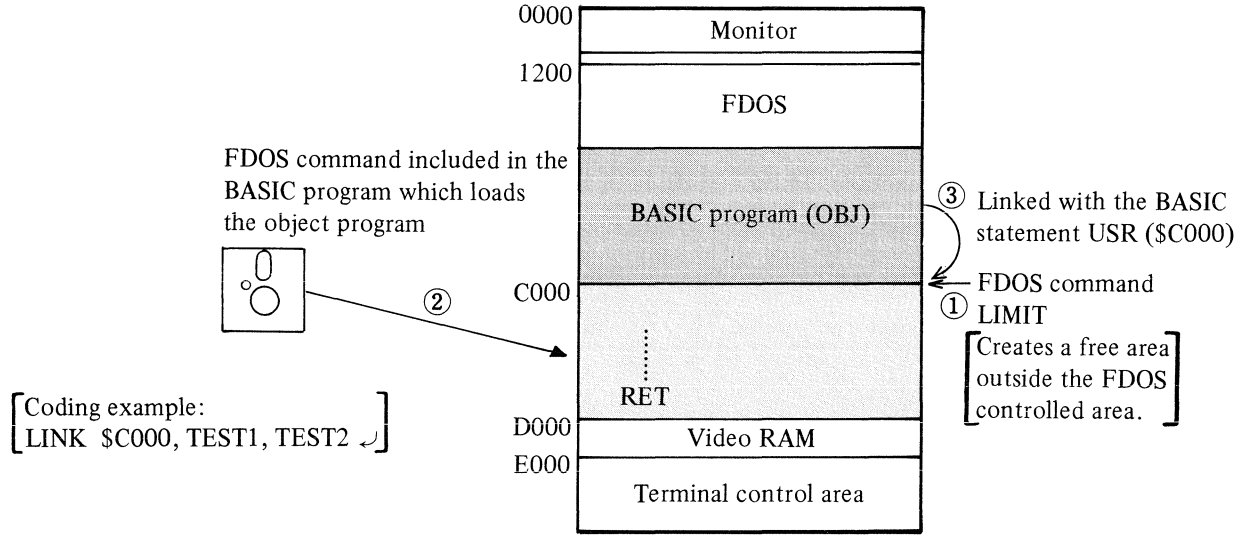
Memory after loading with machine language command Y



Memory after loading with the symbolic debugger

Object programs created with the assembler and BASIC programs created with the BASIC compiler may be linked using a library (see the "Programming Utility" manual) or the BASIC USR statement. Here, an example is given of linking an object program with a BASIC program using the USR statement.

The figure below shows how an object program is loaded and linked with a BASIC program. The area in memory which is managed by FDOS is reduced with the FDOS LIMIT command to create a free area. The object program is loaded into this free area with the BASIC LOAD statement. The BASIC program can then call the object program as a subroutine using the USR( ) statement.



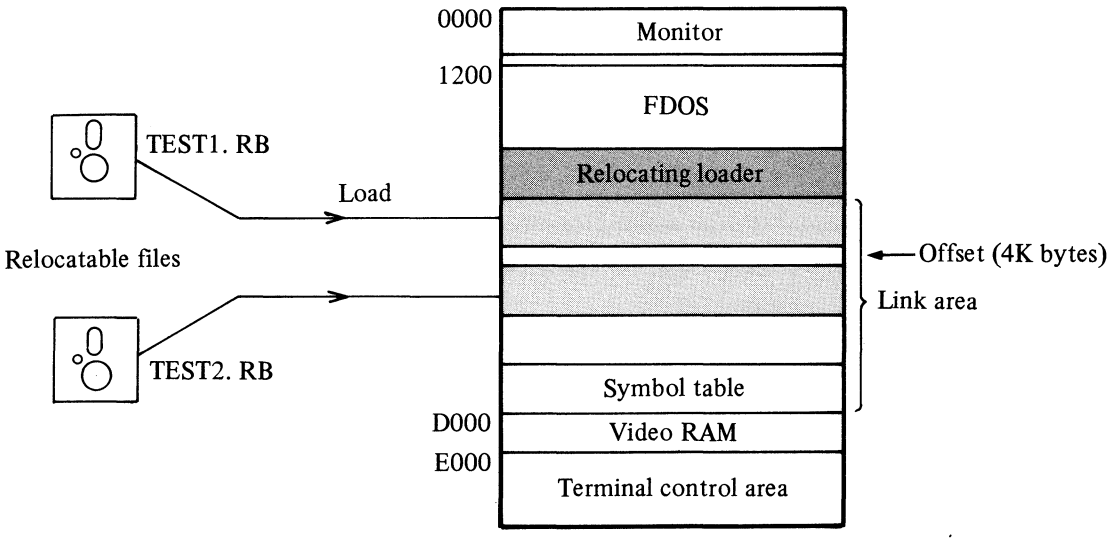
Memory after loading with an FDOS command in a BASIC program

——OFFSET——

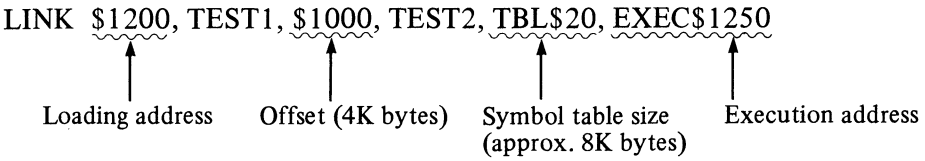
The programmer can specify an offset to reserve a free area between two object program units.

LINK TEST1, $1000, TEST2	Links TEST1 and TEST2 so that the object program is loaded at the area equivalent to 1000 (hex) addresses reserved between them.

Execution of the above command is illustrated below.



Note that the loading address and offset are carefully distinguished in the following command:



A 4-digit hexadecimal number preceded by a $ symbol in the first argument position is always interpreted as the loading address.

——SYMBOL TABLE——

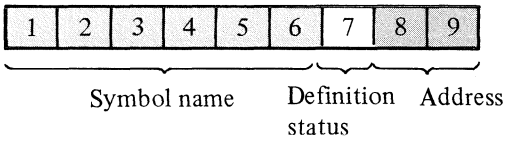
Information referred to as symbols in the relocating loader and symbolic debugger indicates globally declared labels (that is, label symbols defined by the ENT or EQU assembler directive) in the source program. This information is stored in the relocatable file by the assembler for use in linking with other relocatable programs.

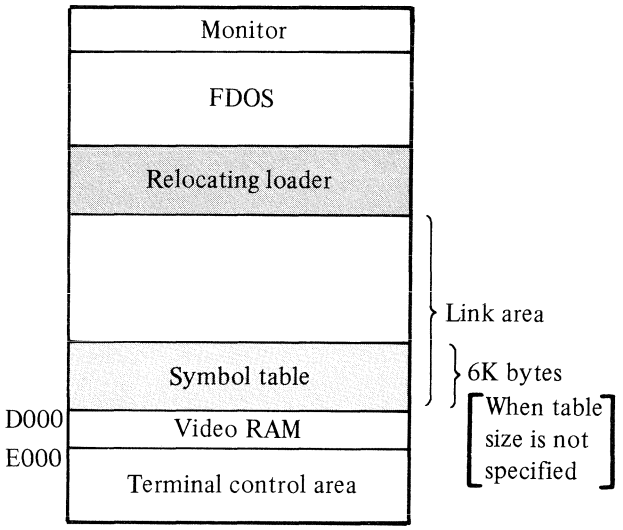
The relocating loader loads label symbols into the symbol table while inputting program units in the relocatable files. The symbol table is placed at the end of the link area; its size is set to approximately 6K bytes by the loader unless otherwise specified by the programmer. The programmer can specify a area of more than 6K bytes for the symbol table area using the LINK command as fillows:

LINK TEST1, TEST2, TBL$20	This command links TEST1 and TEST2 and specifies a symbol table size of 2000H (approximately 8K bytes).

TBL$20 in the above command specifies that a symbol table of approximately 8K bytes is to be created. In other words, the programmer can reserve a symbol table area in 256-byte units. As shown in the memory map, the symbol table is constructed at the end of the link area.

Each symbol table entry is 9 bytes long. The format of the symbol table entry is shown at right. Section 2.3, "Relocating Loader," in the "System Command" manual describes how the loader uses this 9-byte information to link relocatable program units.





Relocating loader memory map

——LINK/T COMMAND——

The LINK/T command is used to display the contents of the symbol table after program linking is completed. It displays a symbol name, its absolute address (in hexadecimal representation) and the definition status for each symbol table entry. The user can detect symbol definition errors by checking the definition status.

The LINK/T command has two basic formats:

LINK/T TEST1, TEST2	Links TEST1 and TEST2 and displays the symbol table on the CRT screen.
LINK/T/P TEST1, TEST2	Links TEST1 and TEST2 and prints the symbol table on the printer.

— The photo at right shows link and symbol table information displayed on the CRT screen with the LINK/T command for the three program units shown on Page 10. Undefined symbols are labeled "U".

— Symbol definition messages are listed below.

Message	Definition
U	Undefined symbol (address or data)
M	Multi-defined symbol (address or data)
X	Cross-defined symbol (address or data)
H	Half-defined symbol (data)
D	Data definition symbol (data)

No message is attached to symbols for which an address has been defined. U, M, X and H indicate error conditions.

The listing below shows a print out of link and symbol table information. The symbol table entries have been sorted as may be seen from this listing.

```
LINKING ROPENTEST.RB
  TOP ASM.BIAS $5600
  END ASM.BIAS $56B1
LINKING RELO.LIB
  TOP ASM.BIAS $56B1
  END ASM.BIAS $67A2
SAVE ROPENTEST.OBJ
  LOADING ADDRESS $5600
  EXECUTE ADDRESS $5600
  BYTESIZE 11A2

SYMBOL TABLE
...ERN  6542     ...N$B  6560  ...SP0  6554  ...T.F  576A
..?SEG  639F     ..CRLF  63F6  ..FLT0  6772  ..INIT  636D
..INT0  5823     ..NARY  56AD  ..NVAR  56AB  ..SEGE  6564
..SEGB  6565     ..TRUE  6529  ..VAR0  56A6  .BRCHK  62CE
.CHKSP  62BF     .DATA0  56A5  .FALSE  652E  .FUNC0  6028
.FUNC1  6043     .G0090  5699  .INPU$  5781  .INPUT  5774
.L000A  560B     .L0014  5623  .L001E  5637  .L0028  564C
.L0032  5662     .L003C  5699  .LINE#  5602  .LOAD$  60B3
EOF     571F     ER14    644D  ER2     6412  ER24    646B
ER3     641F     ER37    6482  ER4     642B  ER6     643D
ER64    672F U   ERR     230B  FASD    5CFC  FBNM    5D9E
FCMP2   5FFD     FCMP5   5FED  FORARR  67A1  FWRK0   61A5
ONERAD  655A     OPNFNC  26B9  OPSP    223F  OUTDEV  6551
OUTIY   6552     PFALSE  5FC9  PNT     654C  PNTDTA  654E
PRNT    0012     PSTACK  6556  PTRUE   5FC4  PUSHR   22F0
PUT1C0  2D30     PUTCR0  2E03  PUTM0   2DD1  REVS    5FB0
RJOB    26DF     ROPEN   22D8  SGETL   28F7  SKPBL   2236
SOUND   2224     STACK$  6558  STAREA  6572  TR5B    619F
XOPEN   22DE     ZMAX    26DD
```

(Note: This listing is not related to the programs on page 10.)

——LINK MESSAGE EXAMPLES——

First program unit loaded (UNIT-#1)

```
TMDLYH :    LD      HL , START
COUNT :     ENT
            DEC     HL
            LD      A , H
            CP      COUNTO
            JR      NZ , COUNT
            LD      A , L
            CP      COUNT1
            JR      NZ , COUNT
            CP      COUNT2
            JR      NZ , COUNT
            RET
PEND :      ENT
            DEFM    'TMDLYH'
            DEFB    0DH
COUNT1 :    EQU     00H
COUNT0 :    EQU     50H
            END
```

Second program unit loaded (UNIT-#2)

```
TMDLYL :    LD      HL , START
LOOP1 :     DEC     H
            LD      A , H
            CP      COUNT
            JR      NZ , LOOP
            RET
PEND :      ENT
            DEFM    'TMDLYL'
            DEFB    0DH
START :     EQU     1000H
COUNT :     EQU     00H

            END
```

Third program unit loaded (UNIT-#3)

```
INPUT :     CALL    001BH
            CALL    TMDLYL
            CALL    001BH
            LD      HL , START
            CP      0DH
            JR      Z , END
            LD      (HL) , A
            INC     HL
            JR      INPUT
END :       JP      0000H
COUNT2 :    EQU     12

            END
```

"START" X

START is not defined as an address in the first program, but is defined as data in the second or subsequent program with the START: EQU statement.

> **NOTE:**
> **The EQU statement should be placed at the beginning of the program unit.**

"COUNT2" H

COUNT2 is not defined as data in the first program, but is defined as data in the third program with the COUNT2: EQU statement.

"COUNT1" D

COUNT1 is defined as data (D indicates no error condition).

"COUNT" X

COUNT is defined as an address in the first program while it is simultaneously defined as data in the second program.

"PEND" M

PEND is defined as an address in the first program while it is simultaneously defined as an address in the second program (duplicated difinition).

"TMDLYL" U

TMDLYL is neither defined as an address nor declared with the ENT directive in any other external program unit.

—— ERROR MESSAGES ——

The error messages issued by the relocating loader are described in the "System Command" manual. Here, only error messages which require particular attention are described.

NO MEMORY SPACE

Indicates that the symbol table is full; that is, that there are too many symbols to be cataloged.

The symbol table size is set to approximately 6K bytes by the relocating loader unless specified by the programmer. It is necessary to specify the TBL$ argument in the LINK command to increase or decrease the symbol table size.

MEMORY PROTECTION

Indicates that the link area is inadequate, that is, that the linked data has reached the symbol table area located at the end of the link area.

IL DATA

Indicates that the data read from the specified relocatable file has an illegal link format. This condition may be caused by a hardware read error in the floppy disk drive or by an assembly error in the source program.

Programming Utility

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

—CONTENTS—

U/PROM

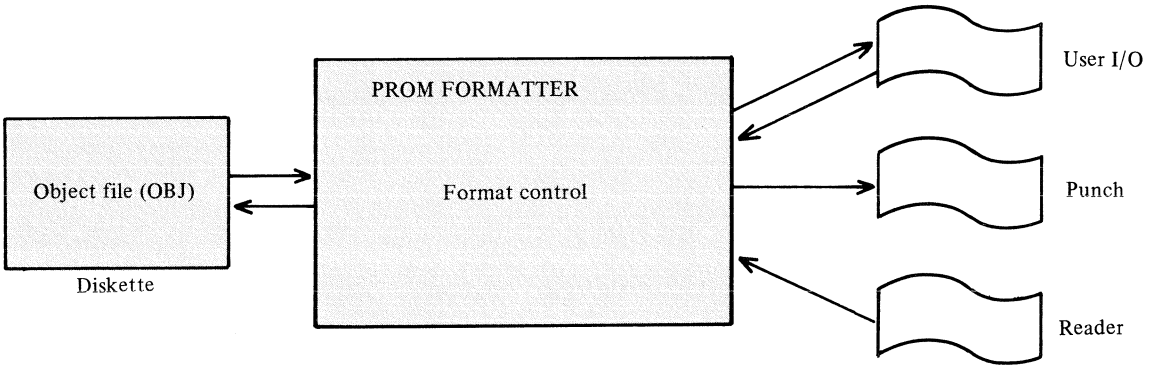
U/PLOTTER

PROM FORMATTER

The rapid advances in LSI technology have allowed the functions of a computer's CPU to be concentrated onto a single semiconductor chip. These microprocessors are becoming ever more sophisticated, while at the same time they are becoming less expensive. As a result, the range of fields in which microprocessors are being utilized is growing rapidly.

One subject of great importance to the development of new device applications is that of developing efficient application programs; it is not too much to say that the quality of the application program determines the how well a newly developed device performs. On the other hand, developments in LSI technology have also stimulated efforts to develop low cost, large capacity memory elements (RAM and ROM). The increased availability of PROMs which are erasable with ultraviolet rays has had a particularly strong influence on the development of devices which incorporate microprocessors.

The procedure which is most suitable for efficiently developing application programs is to create an object file from a source file created through assembly programming using an assembly language, then to finish the program after debugging. The function of the PROM FORMATTER is to load one or more object programs created with an assembler and relocatable loader, then to output it to a paper tape punch after converting it to PROM writer format.



Functions of the PROM FORMATTER

It also allows programs which are written in different formats to be input from a reader for storage on diskette and enables conversion of programs to the required format for output on paper tape or the like.

—— ACTIVATION OF THE PROM FORMATTER ——

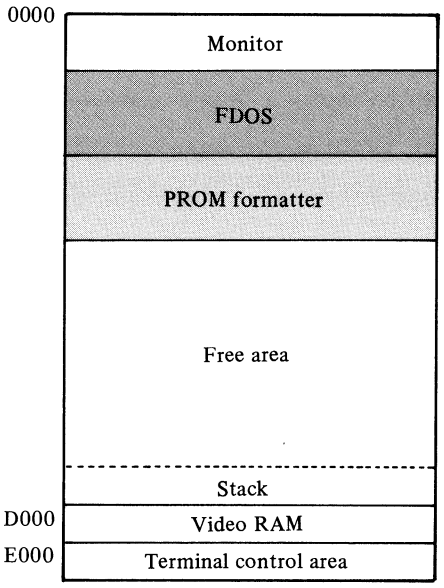
Entering PROM CR while in the FDOS command entry mode activates the PROM FORMATTER. Commands may be entered as soon as activation is completed.

The following formats are provided for in the PROM formatter:

1. **BNPF**
   - Britronics
   - Intel
   - Takeda Riken
2. **B10F**
   (• Takeda Riken)

3. **HEXADECIMAL**
   - Britronics
   - Takeda Riken
   - Minato Electronics

4. **BINARY**
   (• Britronics)



PROM formatter memory map (with 48K bytes of RAM)

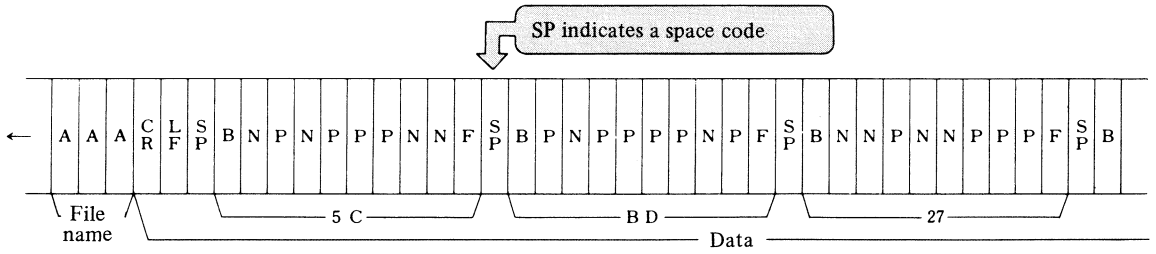
PROM WRITER FORMATS

PROM writers are provided in many formats by different companies. This section discusses forms which are converted by the PROM formatter: refer to the individual PROM writer manuals for details.

The examples in the figures include the file name "AAA," the address "0000," and the data "5C," "BD" and "27." The leader section for the start of punched output and the trailer section for the end of punched output are created automatically.

—— BNPF ——

Britronics (Format A)



—— The file name is punched in ASCII code (if one is specified). (Using the character "B" as a file name will result in incorrect identification of the beginning of data.)

—— CR and LF codes are punched in ASCII code.

—— The space code (20H) and the byte of data at the address specified for RAM FROM? are punched in BNPF format. The address is incremented successively.

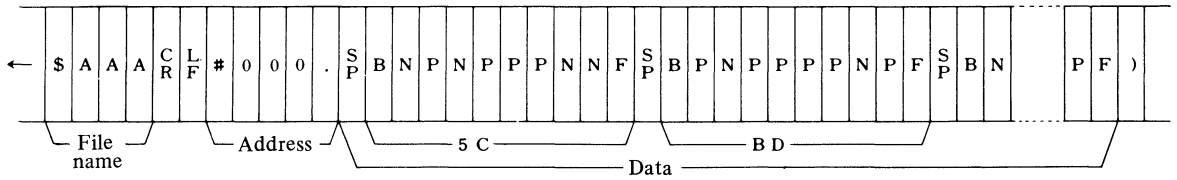
—— CR and LF codes are punched after each 6 items of data are punched in the BNPF format.

—— Punching is performed in BNPF format up to the address specified for TO?

Intel (Format D)

—— This is the same as the Britronics format. The BNPF format is one which has a relatively high degree of standardization; thus, the PROM formatter can also be used with devices other than those which are discussed in this manual.

Takeda Riken (Format E)



—— The "$" mark, which denotes the file name, is punched in ASCII code.

—— The file name is punched in ASCII code (if one is specified).

—— CR and LF codes are punched. The "$" mark is regarded as denoting the beginning of a comment statement; the end of a comment statement is denoted with an LF code.

—— The "#" mark (which indicates the beginning of an address) is punched, followed by the first three digits of the address specified for PROM ADDRESS? The separator between the address and the data is punched as ".".

—— The data item at the address specified for RAM FROM? is punched in BNPF format. The address is incremented successively.

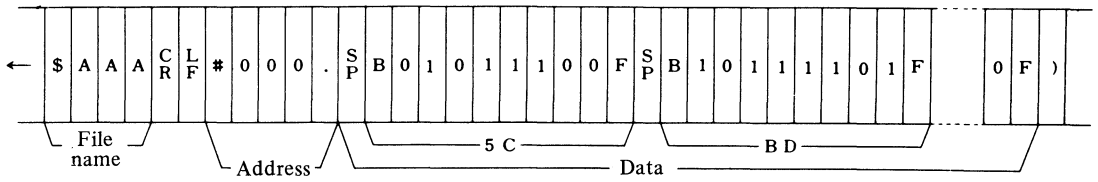
—— CR and LF codes are punched after each 6 items of data are punched in the BNPF format.

—— A tape leader stop mark (")") is punched after the data has been punched up to the address specified for TO?

Note) Care must be taken to ensure that characters which act as control characters (B, :, $, #, etc.) are not used when a file name is specified. (Otherwise, incorrect operation will result.)

—— B10F ——

Takeda Riken (Format F)

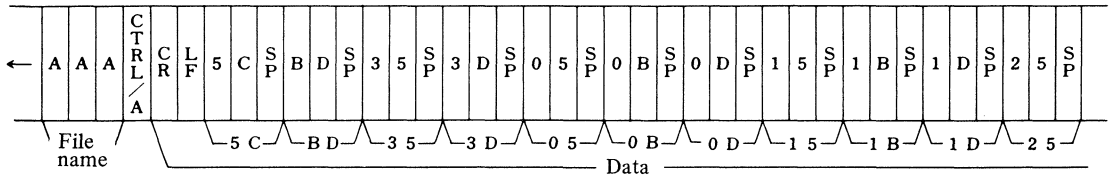


—— Except for the NP section, this is the same as Takeda Riken's BNPF format.

—— The B10F format corresponds to the BNPF format in that 1=P and 0=N.

—— HEXADECIMAL ——

Britronics (Format B)



—— The file name is punched in ASCII code (if one is specified).

—— The "CTRL/A" mark (01H) indicating the beginning of data is punched.

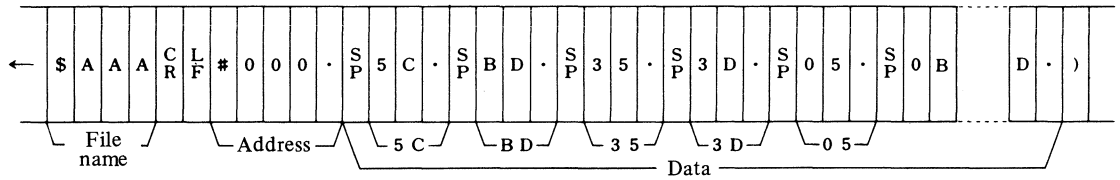
—— CR and LF codes are punched.

—— The data item at the address specified for RAM FROM? is punched as a 2-digit ASCII code, then a space code is punched.

—— CR and LF codes are punched after 16 bytes of data have been punched.

—— Data is punched up to the address specified for TO?

Takeda Riken (Format G)



—— The "$" mark, which denotes the file name, is punched in ASCII code.

—— The file name is punched in ASCII code (if one is specified).

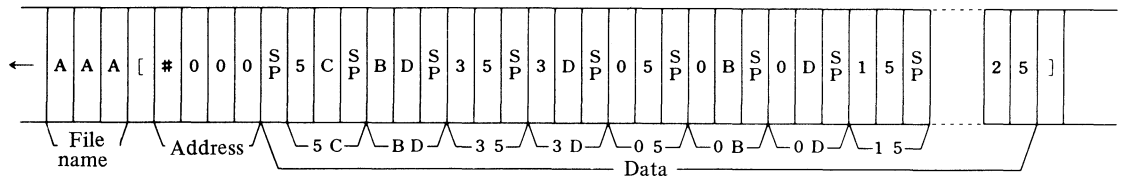
—— After CR and LF codes are punched (followed by the address specification mark "#" and 3 digits of the address specified for PROM ADDRESS?) The separator is "." is punched.

—— The space code is punched, followed by the 2-digit ASCII code for the data at the address specified for RAM FROM? The separator "." is punched after the data item.

—— CR and LF codes are punched after 16 bytes of data have been punched.

—— Data is punched up to the address specified for TO?, at which point the tape leader stop mark (")") is punched.

Minato Electronics (Format H)



—— The file name is punched in ASCII code when filename is specified.

—— The start-of-data mark ("[") is punched.

—— The address designation mark ("#") is punched, followed by a 3-digit ASCII code for the address specified for PROM ADDRESS?

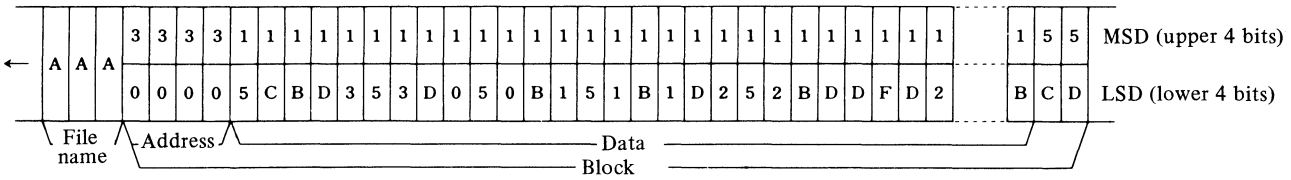
—— A space code is punched, then the data at the address specified for RAM FROM? is converted to a 2-digit ASCII code and punched.

16 combinations of space codes and data items are punched, then CR and LF codes are punched.

—— The end of data mark ("]") is punched after data has been punched up to the address specified for TO?

—— BINARY ——

Britronics (format C)



—— In the binary format, the 4-bit mark section and the 4-bit data section are expressed together as one character (8 bits). The mark section is punched as the upper 4 bits of the paper tape, while the data section is punched as the lower 4 bits.

—— The file name is punched in ASCII code (if one is specified). Specifications which result in a "3" in the upper 4 bits of the ASCII code file name are not permitted. Such specifications will result in incorrect operation, since incorrect determination that the lower 4 bits of the file name are an address will result.

—— Three binary digits for the address specified for PROM ADDRESS? are punched in the lower 4 bits. The address designation mark ("3") is punched in the upper 4 bits.

—— A data mark ("1") is punched in the upper 4 bits and data at the address specified for RAM FROM? is punched in the lower 4 bits.

—— Data is punched 4 bits at a time (with the upper and lower 4 bits punched in alternation) up to the address specified for TO?

—— Check sum marks ("5") are punched in the upper 4 bits, followed in alternation by check sum data in the lower 4 bits.

– PERFORMANCE BOARDS OF VARIOUS COMPANIES –

(Note: Consult the various manufacturers for details.)

a) Intel

2716

2732

8748／8741

3621, 3602, 3622, 3602A, 3622A, 3604, 3624, 3604A, 3624A, 3605, 3625, 3605A, 3625A, 3628, 3608, 3604AL-6, 3604AL

8702A／1702A

8708／8704／2708／2704

8755A

b) Britronics

Company	Element
Intel	3602A／22A, 3604A／24A, 3604AL／24L, 3605／25, 3608／28
Intersil	5600／10, 5603A／23, 5604／24, 5605／25
Fujitsu	7055, 7051, 7052, 7058, 7053, 7059, 7054, 7057
Monolithic Memory	5330／6330, 5331／6331, 5300／6300, 5335／6335, 5336／6336, 5308／6308, 5309／6309, 53134／63134, 53135／63135, 5305／6305, 5306／6306, 53137／63137, 53141／63141, 5340／6340, 5341／6341, 5348／6348, 5349／6349, 5350／6350, 5351／6351, 5352／6352, 5353／6353, 5380／6380, 5381／6381, 5384／6384, 5385／6385, 5386／6386, 5387／6387
Harris	7602／03, 7610A／11A, 7620A／21A, 7640A／41A, 7640AR／41AR, 7642／43, 7644, 7646R／47R, 7648／49, 7608, 7680／81, 7680R／81R, 7680P／81P, 7680RP／81RP, 7683, 7684／85, 7684P／85P, 7686／87, 7686R／87R, 7686P／87P, 7686RP／87RP
Fairchild	93417／27, 93436／36, 93438／48, 93452／52
National Semiconductor	54／74S387, 54／74S287, 54／74S470, 54／74S471, 54／74S570, 54／74S571, 77／87S295, 77／87S296, 54／74S473, 54／74S472, 54／74S572, 54／74S573
Nihon Denki (NEC)	403D, 406D
Raytheon	29660／61, 29600／01, 29612／13
Signetics	82S114／115, 82S126／127, 82S130／131, 82S140／141, 82S136／137, 82S180／181, 82S2708, 82S184／185, 82S190／191
Texas Instruments	54／7488A, 54／74S／88, 54／74S288, 54／74S470, 54／74S71, 54／74S73, 54／74S72, 54／74S75

c) Minato Electronics

Adaptable to all PROMs.

d) Takeda Riken

MOS Type

Element	Bit configuration, capacity (words X bits = capacity)	Maker name											
		Oki Denki	Toshiba	NEC	Hitachi	Fujitsu	Mitsubishi	Intersil	Intel	Texas Instruments	Fairchild	AMD	SIGNE
M O S	256× 8 =2048			μPD454D⑩	HN351702A⑦	MB8503 MB8513	M5L1702⑦		1602A ⑦ 1702A			AM1702A⑦	1702A ⑦
	512× 4 =2048		TMM121C TMM121C-1										
	512× 8 =4096								2704 ⑧				
	1024× 8 =8192	MSM3758⑧	TMM322C⑧	μPD458D⑩	HN462708⑧	MB8518⑧	M5L2708⑧		2708 ⑧ 2758 ⑨	TMS2708⑧	F2708 ⑧	AM2708 ⑧	2708 ⑧
	2048× 8 =16384		TMM323C⑨	μPD2716D⑨	HN462716⑨	MB8516⑨	M5L2716⑨		2716 ⑨	TMS2716⑧ TMS2516⑨			
	4096× 8 =32768								2732	TMS2532			
C M O S	512× 8 =4096							1M6654④					
	1024× 4 =4096							1M6653④					
	1024× 8 =8192						M58460 S						
Compound MOS									8755				

Bipolar Type

Element	Bit configuration, capacity (words X bits = capacity)	Maker											
		Nihon Denki	Hitachi	Fujitsu	Mitsubishi	Intersil	Intel	Texas Instruments	Harris	Fairchild	Raytheon	MMI	AMD	SIGNE
Bipolar	32× 8 =256			MB7051 MB7056	M54730	1M5600 1M5610		SN74188A SN74S188 SN74S288	HM7602 HM76LS03 HM7603			6330-1 6331-1	AM27S18 AM27S19	82S23 82S123
	256× 4 =1024	μPB403D②		MB7052② MB7057②	M54700	1M5603A 1M5623	3601 3621	SN74S287 SN74S387	HM7610A HM7610 HM7611A HM7611	93417 93427	29662 29663 29660 29661	6300-1 6301-1	AM27S20 AM27S21	82S126 82S129
	256× 8 =2048							SN74S470 ③ SN74S471 ③	HM7625R HM7629		29600 29601	6308-1 6309-1 6335-1 6336-1 63135-1		82S114
	512× 4 =2048			MB7053② MB7058②		1M5604 1M5624	3602 3622		HM7620 HM7620A HM7621 HM7621A	93436 93446	29612 29613	6305-1 6306-1	AM27S12 AM27S13	82S130 82S131
	512× 8 =4096	μPB405D μPB425D				1M5605 1M5625	3604A 3624A	SN74S472 SN74S473③ SN74S474③ SN74S475	HM7640 H M7640A HM7641 HM7641A HM7640AR HM7641AR HM7647R HM7648 HM7649	93438 93448		6341-1 6340-1 6348-1 6349-1	AM27S15 AM27S26 AM27S27	82S115 82S140 82S141
	1024× 4 =4096	μPB406D μPB426D⑤		MB7054 MB7059⑤		1M5606 1M5626	3605A 3625A		HM7642 HM7642A HM7643 HM7643A HM7644 HM7644A HM7642P HM7643P	93452 93453		6350-1 6351-1 6352-1 6353-1	AM27S32 AM27S33	82S136 82S137
	1024× 8 =8192	μPB417D μPB427D⑥		MB7055 MB7060⑥			3608A 3628A		HM7680 HM7681 HM7680P HM7681P HM7680R HM7681R HM7680RP HM7681RP HM7608 HM7683	93450 93451		6380-1 6381-1 6384-1 6385-1 6386-1 6387-1 63133-1		82S180 82S181 82S2708
	2048× 4 =8192								HM7684 HM7685 HM7684P HM7685P HM7686 HM7687 HM7686P HM7687P HM7686R HM7687R HM7686RP HM7687RP					82S184 82S185
	2048× 8 =16384						3636		HM7616 HM76160 HM76161					82S190 82S191

○ Elements annotated with the same figures in circles can be used with the same performance boards.

PROM FORMATTER COMMANDS

—— FILE INPUT/OUTPUT COMMANDS ——

Y (Yank File) Command

Reads the object (OBJ) file specified by the file name into the free area.

***YCHARAGEN CR** **FROM ? 8000 TO 87FF**	Reads in CHARAGEN. OBJ Specifies read-in from address 8000 (read up to address 87FF)

—— File name can be specified by entering a Y (Yank file command) when "*" appears to indicate that command entry is awaited.

—— Specify the starting address of the file to be read in as a 4-digit hexadecimal number. (Reading will start at the address specified regardless of the actual data address of the object file.)

—— The last address read is displayed when file read-in is completed.

Caution

The address specified for read-in must be in the free area.

S (Save file) Command

Writes the specified program (or data) in the free area onto a diskette.

***STEST # 2 CR** **FROM ? 8400 TO ? 87E7** **EXECUTE ? 1200 DATA ? 1200**	Output program (data) to TEST# 2.OBJ Output program (data) from address 8400 to 87E7 Execute address 1200, data address 1200

—— File name can be specified by entering an S (Save file) command when "*" appears to indicate that command entry is awaited.

—— The addresses of the memory block in the free area which is to be output are specified with 4-digit hexadecimal numbers.

—— The execute address and data address of the object (OBJ) file created are specified with 4-digit hexadecimal numbers.

The data address is the address to which the program (data) is to be reloaded into memory by a later RUN or LOAD command.

The execute address is the address from which a program reloaded into memory is to be executed.

Specify 0000 when either of these addresses is not necessary.

—— FORMATTING COMMANDS ——

P (Punch) Command

Punches data in the free area in the specified format.

∗P [CR]	Punch command
FILENAME ? CHARAGEN [CR]	File name assigned to the paper tape to be punched.
FORMAT ? C [CR]	Format C
RAM FROM ? 8000 TO ? 87FF	Addresses 8000 to 87FF in the free area
PROM ADRS ? 0000	PROM write address 0000

—— Enter the P (Punch) command when "∗" appears to indicate that command entry is awaited.

—— Next, the file name is specified. This is not the file name which is included on the diskette, but the name which is to be punched at the beginning of the tape. Refer to the explanations of the various formats for details. (When no file name is needed, enter only [CR] .)

—— Next, specify the conversion-format (A-H) and enter [CR] .

—— Specify the starting and ending addresses of the memory block in the free area which is to be output with 4-digit hexadecimal numbers.

—— Finally, specify the PROM write address. (This step may not be required, depending on the format.)

The P command described above outputs formatted data to a PTP device. (More precisely, $PTP/LF is used as the output device.)

The PROM FORMATTER can also output converted format data to devices other than PTP (including user I/O and diskette).

(Ex. 1) ∗P$USR1 [CR] Outputs to user I/O

(Ex. 2) ∗PXYZ [CR] Outputs file name XYZ. ASC to the diskette.

(Ex. 3) ∗P$PTP/PE/LF [CR] Adds even parity to data, affixed [LF] after [CR] and outputs to a PTP device.

An an application, data may be sent directly to the PROM writer by creating hardware and user routines for its online interface.

R (Read) Command

This command reads in data formatted in the BNPF, HEXADECIMAL formats from a paper tape reader.

∗R [CR]	Read command
FORMAT ? C [CR]	Format C
RAM FROM ? 8000 TO 83FF	Addresses in the free area into which data is to
FILENAME PROM# 2	be read.

—— Enter the R (Read) command when "∗" appears to indicate that command entry is awaited.

—— Next, specify the format of the data to be read.

—— Finally, specify the starting address of the free area into which the data is to be read with a 4-digit hexadecimal number.

—— The last data address and the file name are displayed after the read is completed and entry of the next command is awaited.

—— With this PROM writer format, it may not be possible to read tapes punched using other programs because of the need to maintain a certain minimum degree of redundancy.

The R command described above reads in formatted data from a PTR device. (More precisely, $PTR is used as the input device.)

The PROM FORMATTER can also read in converted format data from devices other than PTR (including user I/O and diskette).

(Ex. 1)	∗R$USR2 [CR]	Input from user I/O
(Ex. 2)	∗RXYZ [CR]	Input from XYZ. ASC on a diskette
(Ex. 3)	∗R$PTR/PE[CR]	Inputs data with even parity affixed from PTR.

—— OTHER COMMANDS ——

M (Memory dump/modify) Command

This command is used to display and modify the contents of the free area.

∗M [CR]	M command
RAM FROM? 7000 TO? 7014	Area to be displayed.
7000 ED 73 C8 5F 01 C6 5F CD	
7008 60 22 D8 CD 3E 28 CA 27	
7010 28 22 DE 26 CD	

——Enter the M (Memory dump/modify) command when "*" appears to indicate that command entry is awaited.

——Next, specify the starting and ending addresses in the free area of the data to be displayed with 4 digit hexadecimal numbers.

——The PROM FORMATTER divides data in the specified addresses into 8-byte segments and displays the address (4 hexadecimal digits), the 8 bytes of data (as groups of 2 hexadecimal digits) and the 8 corresponding ASCII characters in that order. However, when the corresponding ASCII character cannot be displayed, a "." is displayed in its place. Further, data is printed in 16 byte segments when the printer is used with the "#" command.

——Execution of the M command can be suspended or resumed by pressing [SPACE]. A switch can be made to the command entry mode by pressing [SHIFT] + [BREAK].

——If no change is required in data displayed using the M command, just press [CR]. When a change is required, move the cursor to the position where the change is to be made and press [CR] after entering the 2 new hexadecimal digits. (The change is made when [CR] is pressed.) After data modification is completed, move the cursor to an empty line and press [CR] to return to the command wait state.

——Data can also be changed using the cursor when display is suspended with [SPACE]. In this case, display is resumed when the cursor is moved to an empty line and [CR] is pressed.

Caution

Data is only printed when the printer is used with the "#" command; modification of data is not possible in this case.

 # (Change printer mode) Command

This command starts and stops output to the printer. Printer output is OFF when the PROM FORMATTER is activated, and is changed from ON to OFF to ON each time the "#" command is executed.

When printer output is ON, data is printed almost as it appears on the display screen.

& (Clear) Command

Buries the entire free area in hexadecimal code (FFH).

? (Display free area) Command

Displays the free area.

! (Return) Command

Terminates the PROM FORMATTER and returns to FDOS.

—— FORMAT COMMANDS ——

Format commands are commands entered when "FORMAT?" is displayed during execution of the P and R commands. Selecting one of these commands during execution of the P command determines whether data is to be punched in BNPF, HEXADECIMAL or other format. Failure to specify the correct format command during execution of the R command will result in failure to correctly read the program into the free area.

A Command

——Used to specify the Britronics BNPF format. The control character "B" may not be used when the file name is specified.

B Command

——Used to specify the Britronics HEXADECIMAL format.

C Command

——Used to specify the Britronics BINARY format. Numerals and the codes (: ; < = > ?) may not be used when the file name is specified.

——The message "PROM ADDRESS?" is displayed during execution of the P command to request specification of the PROM loading address; specify it as a 4-digit number.

——Check sums are written following the data (with the P command).

——Data from the address specification to the check sums constitutes one block; if data is to be loaded into an address which has been skipped, the operation must be divided into two or more parts. This also applies when two or more blocks are read in with the R command.

D Command

——This command is used to specify the Intel BNPF format.

—— The character "B" cannot be used in the file name.

E Command

—— This command is used to specify the Takeda Riken BNPF format. The character "B" may be used in the file name.

—— A file is a block which begins with "$" and ends with ")".

—— The message "PROM ADDRESS?" is displayed during execution of the P command to request specification of the PROM loading address; specify it as a 4-digit number.

—— If two or more blocks are to be read out or written in, the operation must be divided into two or more parts.

F Command

—— This command is used to specify the Takeda Riken B10F format. The character "B" may be used in the file name.

—— A file is a block which begins with "$" and ends with ")".

—— The message "PROM ADDRESS?" is displayed during execution of the P command to request specification of the PROM Loading address; specify it as a 4-digit number.

—— If two or more blocks are to be read out or written in, the operation must be divided into two or more parts.

G Command

—— This command is used to specify the Takeda Riken HEXADECIMAL format.

—— A file is a block which begins with "$" and ends with ")".

—— The message "PROM ADDRESS?" is displayed during execution of the P command to request specification of the PROM loading address; specify it as a 4-digit number.

—— If two or more blocks are to be read out or written in, the operation must be divided into two or more parts.

H Command

—— This command is used to specify the Minato Electronics HEXADECIMAL format.

—— The start-of-data symbol ("[") may not be used in the file name.

—— The message "PROM ADDRESS?" is displayed during execution of the R command to request specification of the PROM loading address; specify it as a 4-digit number.

—— Denote the end of data with the symbol "]".

DIFFERENCES BETWEEN PROM FORMATTERS SP-7501 AND SP-2501

This manual is written for FDOS PROM formatter SP-7501; the principal differences between this version and the previous cassette-based version (PROM formatter SP-2501) are described below for reference.

Item	SP-2501	SP-7501
Principal functions	This version includes a RELOCATING LOADER (SP-2301) and SYMBOLIC DEBUGGER (SP-2401) as well as the PROM formatter. These allow input of relocatable files (RB). File input/output is from/to cassette tape only.	The OBJ file must be created using the FDOS LINK command. Object file I/O is from/to diskette only. I/O for converted format files (with the exception of $MEM and $CMT) is possible with all FDOS devices.
Parity specification	Paper tape parity is determined with the ＊FPC (Parity Change) command.	Non-parity is standard; if a parity specification is to be made, specify it in the following manner: ＊P$PTP/PE/LF.
Commands and messages	Some of the commands and messages are different.	

PROM FORMATTER (SP-7501) COMMANDS & MESSAGES

COMMAND		OPERATION
File Input/ Output Commands	S (Save)	Saves the program (data) in the free area on diskette.
	Y (Yank)	Loads a program (data) from the diskette into the free area.
Format commands	P (Punch)	Punches the specified contents of the free area in the specified format.
	R (Read)	Reads in a paper tape punched in the format specified in the command.
Other commands	M (Memory)	Displays and modifies data in the free area.
	& (Clear)	Buries all data in the free area in hexadecimal code FFH.
	#	Switches the list mode for listing on a printer.
	?	Displays the starting and ending addresses of the free area.
	! (Return)	Returns control to FDOS.

Error message	Error content	Related commands
MEMORY PROTECTION	An address outside of the free area was specified.	Y, S, P, R, M
IL COMMAND	The command was not entered correctly.	Y, S, P, R
IL DATA	The format specified does not match the format read.	R
CHECK SUM	Check sum error	R
$LPT:NOT READY	The printer is not ready.	#
$PTP:NOT READY	The paper tape punch is not ready.	P
$PTR:NOT READY	The paper tape reader is not ready.	R

See the "System Error Messages" in System Command for other error messages.

Note:

Entry of characters other than S Y P R M & # ? or ! will cause a return to the command wait state after the command table is displayed.

If a character other than A-H is input while FORMAT? is displayed and format entry awaited, the format table will be displayed, after which the format entry wait state will be reentered. A return can be made to the command wait state at this time by pressing [SHIFT] + [BREAK].

EXAMPLE OF PLOTTER CONTROL APPLICATION

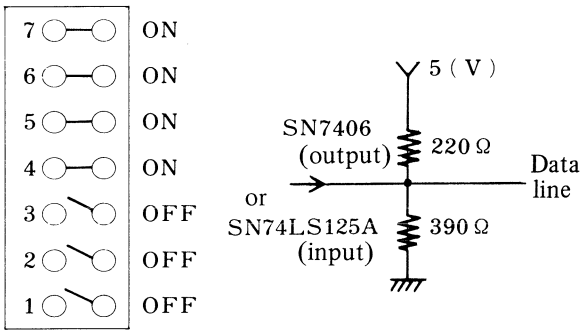
Using the eidtor, assembler and BASIC compiler of FDOS, main programs can be written in the familiar BASIC language without reduction in processing speed if control programs are made for control hardware. Another benefit is that commands developed by the user can be used as BASIC commands.

—— INTERFACE CARD ——

The universal interface card MZ-80I/O-1 is used for the interface between the MZ-80 series computer and the MIPLOT WX4671 plotter.

The connection conditions are as hown in Figure 1.

Output port (0FH)		Input port (0EH)	
I/O card	Plotter side	I/O card	Plotter side
O 27	STROBE	I 17	Grounded on the universal interface card
O 26	DB 6	I 16	
O 25	DB 5	I 15	
O 24	DB 4	I 14	
O 23	DB 3	I 13	
O 22	DB 2	I 12	
O 21	DB 1	I 11	ERROR
O 20	DB 0	I 10	BUSY



a) Connection conditions for all input terminals **b) Address switch settings** **c) Data line termination conditions**

Fig. 1 Connection conditions

SN7404 is included as the data driver for the universal interface card, but ICs 14 and 15 only are changed to SN7406. All data line and status input terminations are made as shown in Figure 1-c). A 1.5m cable can be used for this purpose.

See the universal interface card instructions for details.

—— PLOTTER CONTROL PROGRAM ——

This section may skip if you are not interested in assembly subroutines.

1. Conditions for linkage with a BASIC statement

Conditions for linkage with a BASIC statement

(a) Command names must be externally declared with the ENT statement .....SIZE:ENT

(b) The number of parameters must be specified. ..... DEFB 1 (1 parameter)

(c) The parameter type must be specified. ..... DEFB 0 (real number)

(d) Buffers must be specified for parameters ..... SE:DEFS 2 (2 bytes reserved)

(e) The RET instruction must be included at the end of all control routines.

The above are the linkage conditions; the processing program is written between items (d) and (e).

2. Linkage conditions when an error occurs

(a) A subroutine is used from FDOS library RELO. LIB. ..... CALL BEERR
(b) The error number is written ..... DEFB 80
(c) The error message is written ..... DEFM 'PLOTTER ERROR'
(d) The terminator is written ..... DEFB 0DH

This causes *ER 80: PLOTTER ERROR to be output on the display screen when a plotter error occurs.

3. Use of external subroutines

This control program uses 4 routines out of the subroutines included in FDOS library RELO. LIB. One of these is BEERR, which was shown above; the remaining three are as follows:

(a) ..INT 0

16 bit binary data is set in the HL register with a sign attached when an address with a parameter is loaded in the HL register and called. All registers except the AF register are protected in the event of an overflow if the carry flag is set.

(b) CASC'

The unsigned 16-bit binary data from the HL register is converted to ASCII code and stored in the address indicated in the DE register, then ODH is set.

(c) .MOVE'

When the parameter contains a type 1 string and an address with data is loaded in the HL register and called, a type 2 string is set in the address indicated by the DE register and 0DH is set.

See the "Library/Package" instructions for details on all subroutines.

4. Plotter control codes

All data for the MIPLOT WX4671 currently used is in 7-bit ASCII code. Input statuses include the BUSY signal and the ERROR signal. Data output is possible when the BUSY signal goes low, and the data is taken in on the plotter side when the STROBE signal is output.

"," (that is, 2CH) is used as the data delimiter and 03H-0DH are valid as data terminators. However, only 0AH will be accepted when an error occurs; an error condition is not cleared by any data terminator other than 0AH.

At the port on the MZ-80 side, 0EH is used for the status (that is, as the input for the BUSY and ERROR signals) and 0FH is used for the data and STROBE signal output.

5. Program outline

Linkage conditions for a BASIC compiler have already been indicated; however, string type becomes applicable in the parameter type specification with the 80H. Moreover, parameter types and parameter buffers must be added depending on the number of parameters; the steps described in subparagraphs (c) and (d) of that section are not required if the number of parameters is zero. See routines CTYPE, SIZE, PLOT, HOME and so forth of the assembly listing for this.

This is illustrated using the PLOT routine as a representative example. The flowchart is as shown in Figure 2 (pages 4 and 5).

(a) Data output

Although subroutines COUT, PLOT1 and DOUT are used, DOUT is the one which actually outputs the data. With DOUT, the data set in the accumulator is output to the plotter with the STROBE signal if the BUSY signal of the plotter is LOW. If BUSY is HIGH the routine repeats a loop.

With COUT and PLOT1, the data at the address indicated in the HL register is loaded in the accumulator and then DOUT is executed; this is repeated until 0DH is output. After 0DH is output, a check is made for plotter errors and a jump is made to the error routine if any are found. Note that continuation of program execution is possible if ON ERROR processing is provided on the BASIC side.

(b) Data conversion routine

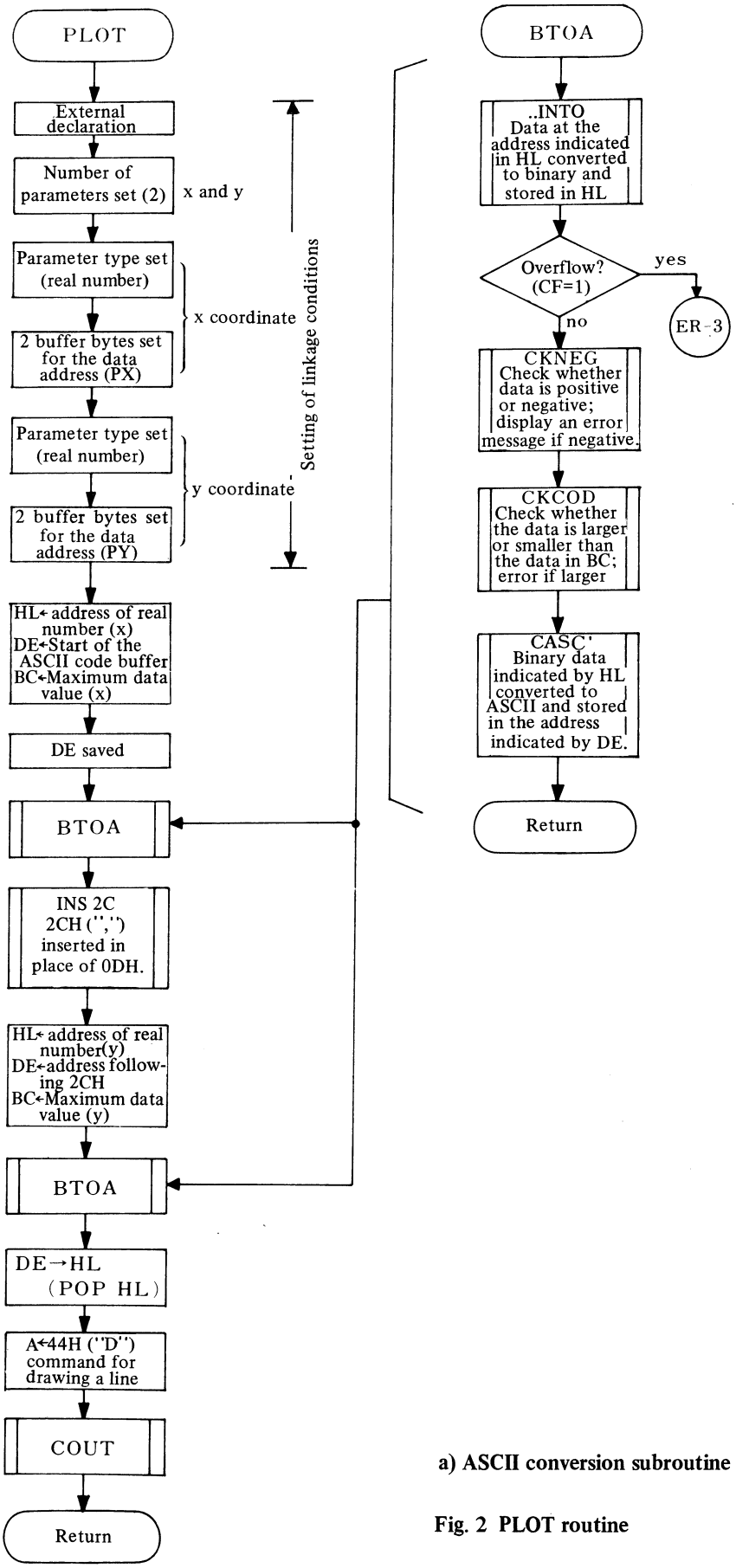
As was indicated previously, all data must be converted into ASCII code since the plotter will not accept data in any other form.

Subroutine BTOA uses the RELO.LIB routine in FDOS to convert data to ASCII code. This is as shown in the flowchart in Figure 2-a) and paragraph 3, "Use of external subroutines"; however, checks are also made for positive, negative and overflow data.

(c) PLOT x, y routine

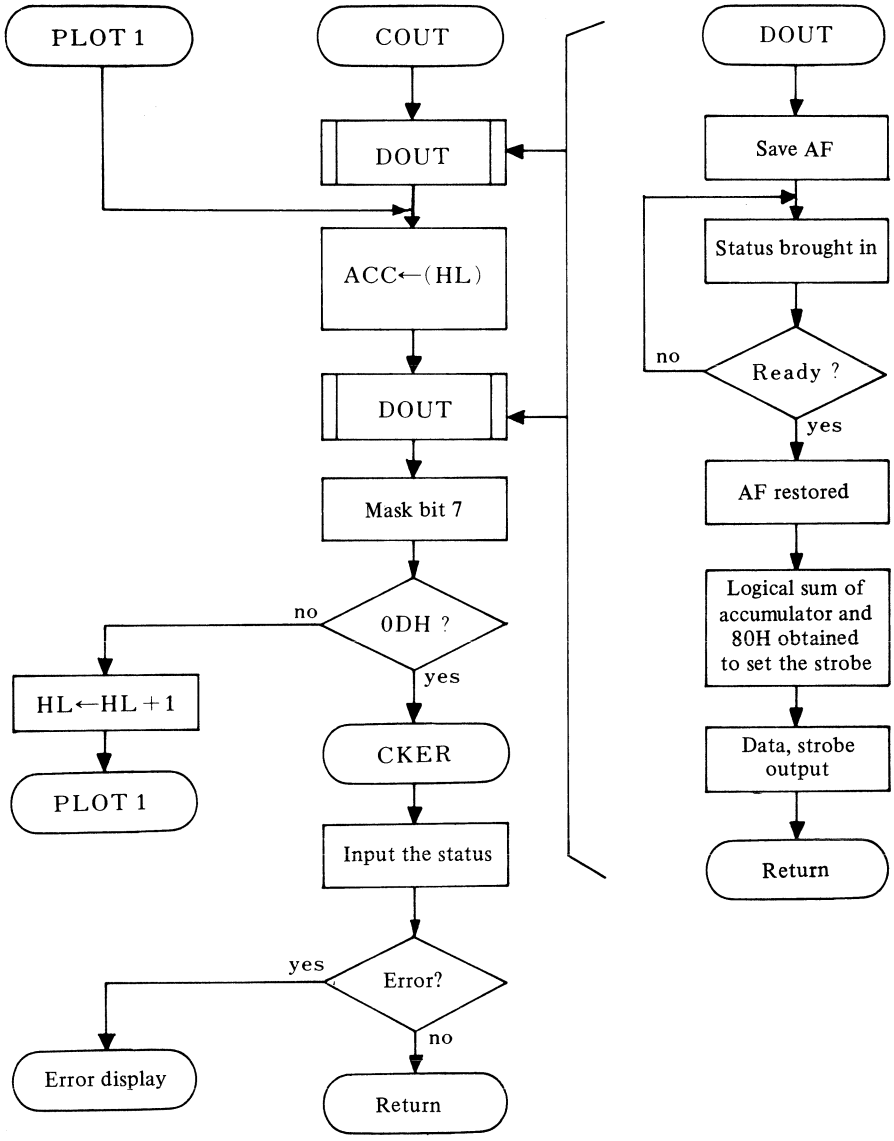
As is shown in the flowchart, the two parameters x and y are specified following the external declaration. The parameter type is real number and two bytes are set in the parameter buffer for both x and y.

The pre-conversion data address, the starting address of the buffer for storage of the data after conversion and the maximum value of x are loaded in registers HL, DE and BC, respectively, then BTOA is called and the data is checked and converted to ASCII code. Afterwards, the data delimiter for the plotter (",") is loaded and the process is repeated for y. The starting address of the data converted into ASCII code is loaded into the HL register, the 44H ("D") command (which draws a line on the plotter) is loaded and COUT is called. With COUT, the contents of the address indicated in the HL register are output to the plotter following the command.



a) ASCII conversion subroutine

Fig. 2 PLOT routine



b) Data output subroutine

Fig. 2 PLOT routine

—— COMMAND TABLE ——

Table 1 Plotter control commands

Command name	Function
PLOT x, y	Draws a straight line from the current position of the pen to the coordinates x, y; x = 0–3600, y = 0–2540 are the possible range (other values will result in an error).
IPLOT Δx, Δy	Draws a straight line from the current position of the pen for the values of ±Δx, ±Δy only.
MOVE x, y	Lifts the pen and moves it to the position indicated by coordinates x and y. x = 0-3600, y = 0-2540 are the possible range (other values will result in an error).
IMOVE Δx, Δy	Lifts the pen and moves it by an amount indicated by ±Δx, ±Δy.
XAXIS x, n	Draws a scale with n divisions on the X axis at intervals indicated by x.
YAXIS x, n	Draws a scale with n division on the Y axis at intervals indicated by y.
CTYPE A$	Prints the character string indicated by A$.
SIZE n	Specifies that characters are to be written in the size indicated by n; n = 0–15.
CSIZE	Specifies that characters are to be written in the size initially set for the plotter (n = 4 is output automatically).
DOT x	Draws a dotted line over the interval specified by x (x≤127).
DOTM	Draws a line of dots spaced at intervals of 3 mm.
LINE	Clears DOT x or DOTM and returns to a straight line.
PUP	Pen up (same function as IMOVE0, 0)
PDOWN	Pen down (same function as IPLOT0, 0)
ANGL n	Rotates characters by the angle (in the counterclockwise direction) specified by n. n = 0–3 n = 0...0° n = 1...90° n = 2...180° n = 3...270°
MARK n	Writes the mark specified by n. n = 1–6 n = 1 ... . n = 2 ... ◇ n = 3 ... ▣ n = 4 ... △ n = 5 ... ⧖ n = 6 ... ⬡
HOME	Moves the pen to the x, y coordinates it was at when the power was turned on clears the error lamp and clears plotter errors.
ERCLR	Clears plotter errors when they occur; the error lamp is not cleared, however.
(Note)	Values indicated by x and y are specified as integral multiples of 0.1 mm; for example, PLOT 100,2000 results in a line being drawn to x = 10 mm and y = 200 mm.

—— OUTLINE OF THE BASIC PROGRAM ——

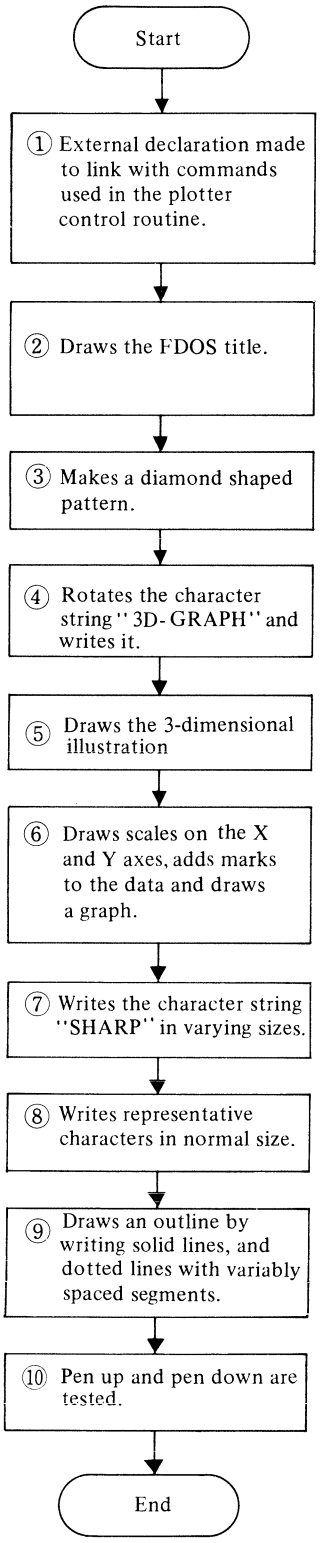


Fig. 3 Flowchart of the BASIC program

As is indicated in the flowchart at left, the program consists of 10 subroutines.

At ①, the EXTERNAL statement is used for linking the program with the plotter control program commands. Although it is not necessary to use line numbers with statements other than GOTO and GOSUB, they are used in other locations to make the program easier to understand.

At ②, the title of FDOS is drawn from line number 110 to 910. This routine uses the MOVE, IMOVE, PLOT and IPLOT commands.

At ③, a diamond pattern is drawn from line number 920 to 1050. MOVE X, Y is used to return the pen to the starting point, while IPLOT A, B is used to draw the pattern.

At ④, "3D-GRAPH" specified by A$ is written at 90° angles with ANGL I and CTYPE A$ from line number 1090 to 1140.

At ⑤, repetitions of attenuated SIN (X) and a 3 dimensional graph are drawn from line number 1190 to 1910. The 3 dimensional graph is drawn in dots using the PUP and PDOWN commands.
Try changing the program from line number 1500 to 1510 as shown below to draw the graph with solid lines.

```
1500  IF (A-2) > T THEN 1520
1502  T=A
1504  IF S=0 THEN MOVE DX, DY:GOTO 1508
1506  PLOT DX, DY
1508  PDOWN:S=1
1510  RETURN
1520  MOVE DX, DY:T=A:S=0:GOTO 1510
```

With this program, S indicates the pen status; s=0 raises the pen while s=1 lowers it.

At ⑥, scales are drawn on the X and Y axes with the XAXIS and YAXIS commands and marks are drawn on the graph with the MARK command; this processing is performed from line 1990 to line 2190. The dotted line is drawn with the DOTM command, and the LINE command is used to return to a solid line. The curve is a drawing of SIN (X)/X. Refer to the command table for these commands.

At ⑦, the SIZE command is used from line number 2390 to 2460 to write the word SHARP in characters of varying sizes. Afterwards, the CSIZE command is used to return to standard size.

At ⑧, representative characters of standard size are drawn from line number 2790 to 2820.

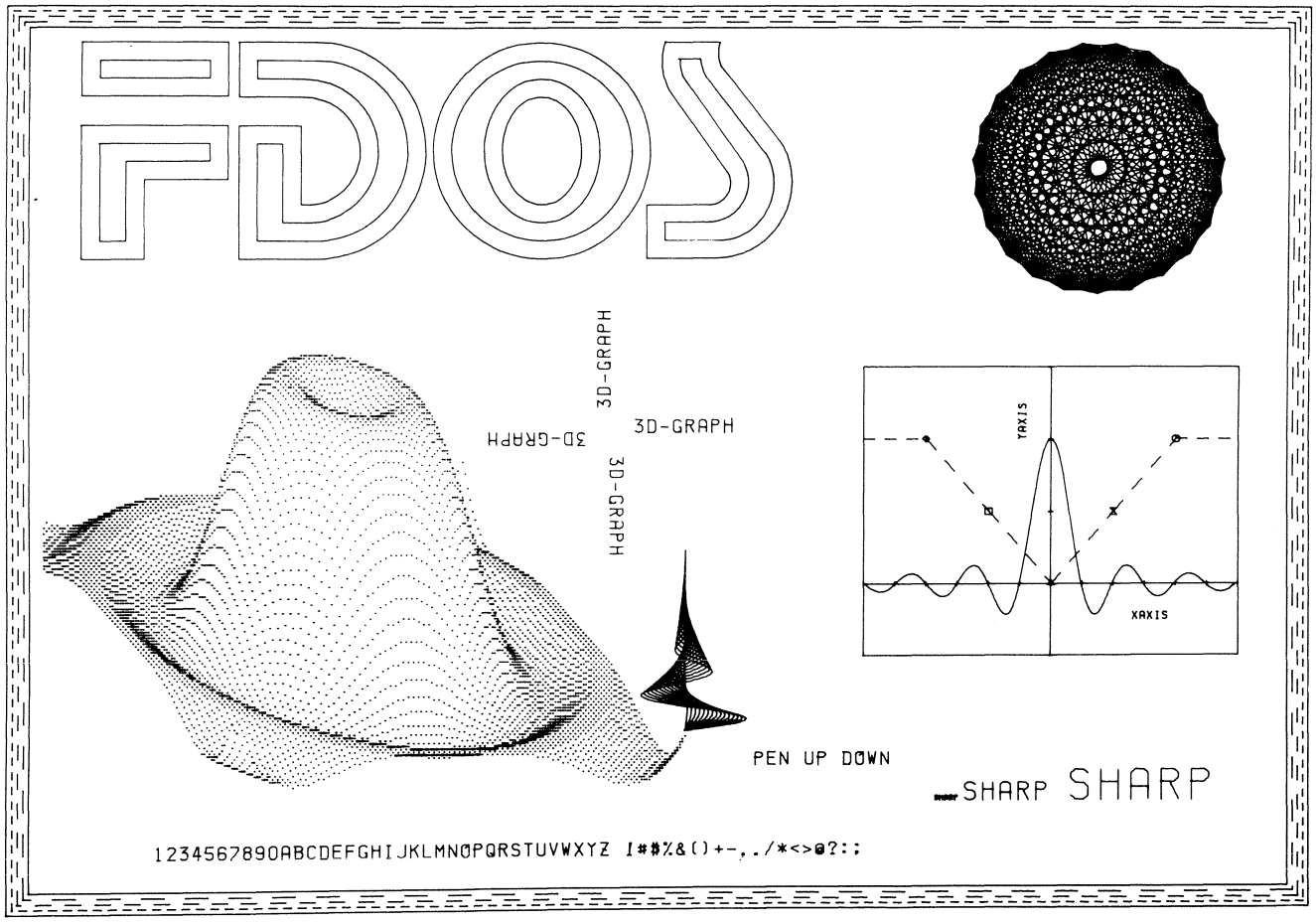
At ⑨, the HOME, DOT and LINE commands are used from line number 3000 to 5010 to draw the surrounding outline by changing between solid lines and dotted lines of varying pitch.

At ⑩, pen up and pen down are tested from line number 5100 to END.

If the program is written so that the HOME command can be executed if an error occurs with the plotter, the error can be cleared, the error lamp extinguished and the pen returned to the starting position. If the ERCLR command is used (it is not in this program), the error can be cleared without moving the pen; however, in this case the error lamp is not extinguished.

—— Conclusion ——

As has been shown above, user-written programs can be easily linked with BASIC. By observing the conditions for linkage, it is possible to connect many devices other than the plotter. When a main program is created using the assembler, it may be necessary to output a picture on the display screen. The BASIC compiler and FDOS are very useful for this purpose. Further, the processing is the same as with machine language. The photograph on page 9 shows the result of execution of this program.



References

1. Watanabe Manufacturing Co., MIPLOT WX4671 Instruction Manual

2. Sharp Corp., Universal Interface Card Instruction Manual

——SAMPLE PROGRAM (PLOTTER CONTROL ROUTINES)——

```
   **  Z80 ASSEMBLER SP-7101  PAGE 01  **

01 0000                    ;;;;;;;;;;;;;;;;;;;
02 0000                    ;
03 0000                    ; PLOTTER CONTROL
04 0000                    ;
05 0000                    ;   PLOTTER WX 4671
06 0000                    ;   UNIVERSAL I/O
07 0000                    ;
08 0000                    ;;;;;;;;;;;;;;;;;;;
09 0000                    ;
10 0000                    ; BASIC SUBROUTIN LINK
11 0000                    ;
12 0000                    ;
13 0000                    ; CALL ..INT0
14 0000                    ;         WITHOUT AF KEEP
15 0000                    ;         16BIT BINARY TO HL
16 0000                    ;
17 0000                    ; CALL CASC'
18 0000                    ;         WITHOUT HL&AF KEEP
19 0000                    ;         HL(16BIT BINARY)TO DE ADDRESS
20 0000                    ;         (DE)=ASCII END=0DH
21 0000                    ;
22 0000                    ; CALL BEERR
23 0000                    ;         ERROR MSG DSP-OUT
24 0000                    ;
25 0000                    ; CALL .MOVE'
26 0000                    ;         CHR TYPE PARAMETER TO ASCIICODE(HL TO DE)
27 0000                    ;
28 0000                    ;
29 0000                    ; PORT
30 0000                    ;
31 0000  P                 PLTS:   EQU   0EH              ;PLOTTER STATUS READ
32 0000  P                 PLTD:   EQU   0FH              ;PLOTTER DATA OUT
33 0000                    ;
34 0000                    ; MOVE X,Y(PEN UP)
35 0000                    ;
36 0000                    MBUFF:  DEFS  +16
37 0010                    ;
38 0010                    MOVE:   ENT
39 0010 02                         DEFB  2                ;NO OF PARAMETER
40 0011 00                         DEFB  0                ;REAL NO.
41 0012                    MX:     DEFS  +2
42 0014 00                         DEFB  0                ;REAL NO.
43 0015                    MY:     DEFS  +2
44 0017                    ;
45 0017 2A1200                     LD    HL,(MX)
46 001A 110000                     LD    DE,MBUFF
47 001D D5                         PUSH  DE
48 001E 01110E                     LD    BC,0E11H         ;3600
49 0021 CD3C02                     CALL  BTOA
50 0024 CD6D02                     CALL  INS2C
```

```
0027 2A1500                     LD    HL,(MY)
002A 01ED09                     LD    BC,09EDH          ;2540
002D CD3C02                     CALL  BTOA
0030 E1                         POP   HL
0031 3E4D                       LD    A,4DH             ;'M' MOVE CMD
0033 C34201                     JP    COUT
0036                    ;
0036                    ; INCREASE MOVE(PEN UP)DX,DY
0036                    ;
0036                    IMBUFF: DEFS  +16
0046                    ;
0046                    IMOVE:  ENT
0046 02                         DEFB  2                 ;NO OF PARAMETER
0047 00                         DEFB  0                 ;REAL
0048                    IMX:    DEFS  +2
004A 00                         DEFB  0                 ;REAL
004B                    IMY:    DEFS  +2
004D                    ;
004D 2A4800                     LD    HL,(IMX)
0050 113600                     LD    DE,IMBUFF
0053 D5                         PUSH  DE
0054 01110E                     LD    BC,0E11H          ;3600
0057 CD4C02                     CALL  NSBTOA
005A CD6D02                     CALL  INS2C
005D 2A4B00                     LD    HL,(IMY)
0060 01ED09                     LD    BC,09EDH          ;2540
0063 CD4C02                     CALL  NSBTOA
0066 E1                         POP   HL
0067 3E52                       LD    A,52H             ;'R' RELATIVE MOVE CMD
0069 C34201                     JP    COUT
006C                    ;
006C                    ; X-AXIS
006C                    ;
006C                    QRXBUF: DEFS  +16
007C                    ;
007C                    XAXIS:  ENT
007C 02                         DEFB  2                 ;NO OF PARAMETER
007D 00                         DEFB  0                 ;REAL
007E                    QX:     DEFS  +2
0080 00                         DEFB  0                 ;REAL
0081                    RX:     DEFS  +2
0083                    ;
0083 2A7E00                     LD    HL,(QX)
0086 116C00                     LD    DE,QRXBUF
0089 D5                         PUSH  DE
008A 01110E                     LD    BC,0E11H          ;3600
008D CD3C02                     CALL  BTOA
0090 CD6D02                     CALL  INS2C
0093 2A8100                     LD    HL,(RX)
0096 01110E                     LD    BC,0E11H          ;3600
```

```
0099 CD3C02                     CALL  BTOA
009C 3E31                       LD    A,31H             ;'1'
009E 1832                       JR    AXOUT
00A0                    ;
00A0                    ; Y-AXIS
00A0                    ;
00A0                    QRVBUF: DEFS  +16
00B0                    ;
00B0                    YAXIS:  ENT
00B0 02                         DEFB  2                 ;NO OF PARAMETER
00B1 00                         DEFB  0                 ;REAL
00B2                    QV:     DEFS  +2
00B4 00                         DEFB  0                 ;REAL
00B5                    RV:     DEFS  +2
00B7                    ;
00B7 2AB200                     LD    HL,(QV)
00BA 11A000                     LD    DE,QRVBUF
00BD D5                         PUSH  DE
00BE 01ED09                     LD    BC,09EDH          ;2540
00C1 CD3C02                     CALL  BTOA
00C4 CD6D02                     CALL  INS2C
00C7 2AB500                     LD    HL,(RV)
00CA 01ED09                     LD    BC,09EDH          ;2540
00CD CD3C02                     CALL  BTOA
00D0 3E30                       LD    A,30H             ;'0'
00D2 CD2002             AXOUT:  CALL  AXIS
00D5 E1                         POP   HL
00D6 186D                       JR    PLOT1
00D8                    ;
00D8                    ; INCREASE PLOT X+DX,Y+DY
00D8                    ;
00D8                    IPBUFF: DEFS  +16
00E8                    ;
00E8                    IPLOT:  ENT
00E8 02                         DEFB  2                 ;NO OF PARAMETER
00E9 00                         DEFB  0                 ;REAL
00EA                    IPX:    DEFS  +2
00EC 00                         DEFB  0                 ;REAL
00ED                    IPY:    DEFS  +2
00EF                    ;
00EF 2AEA00                     LD    HL,(IPX)
00F2 11D800                     LD    DE,IPBUFF
00F5 D5                         PUSH  DE
00F6 01110E                     LD    BC,0E11H          ;3600
00F9 CD4C02                     CALL  NSBTOA
00FC CD6D02                     CALL  INS2C
00FF 2AED00                     LD    HL,(IPY)
0102 01ED09                     LD    BC,09EDH          ;2540
0105 CD4C02                     CALL  NSBTOA
0108 E1                         POP   HL
```

```
0109 3E49                       LD    A,49H             ;'I' INCREASE CMD
010B 1835                       JR    COUT
010D                    ;
010D                    ; PLOT X,Y
010D                    ;
010D                    PTBUFF: DEFS  +16
011D                    ;
011D                    ;
011D                    PLOT:   ENT
011D 02                         DEFB  2                 ;NO OF PARAMETER
011E 00                         DEFB  0                 ;REAL
011F                    PX:     DEFS  +2
0121 00                         DEFB  0                 ;REAL
0122                    PY:     DEFS  +2
0124                    ;
0124 2A1F01                     LD    HL,(PX)
0127 110D01                     LD    DE,PTBUFF
012A D5                         PUSH  DE
012B 01110E                     LD    BC,0E11H          ;3600
012E CD3C02                     CALL  BTOA
0131 CD6D02                     CALL  INS2C
0134 2A2201                     LD    HL,(PY)
0137 01ED09                     LD    BC,09EDH          ;2540
013A CD3C02                     CALL  BTOA
013D E1                         POP   HL
013E 3E44                       LD    A,44H             ;'D' DRAW CMD
0140 1800                       JR    COUT
0142                    ;
0142                    ; ASCII CODE OUT SUB
0142                    ;
0142 CD2C02             COUT:   CALL  DOUT
0145 7E                 PLOT1:  LD    A,(HL)
0146 CD2C02                     CALL  DOUT
0149 E67F                       AND   7FH
014B FE0D                       CP    0DH
014D 2803                       JR    Z,CKER
014F 23                         INC   HL
0150 18F3                       JR    PLOT1
0152                    ;
0152                    ; ERROR CHECK
0152                    ;
0152 DB0E               CKER:   IN    A,(PLTS)
0154 E602                       AND   02H               ;CKER(BIT 1=0?)
0156 C0                         RET   NZ
0157 CD0000     E       CKER1:  CALL  BEERR
015A 50                         DEFB  50H               ;80
015B 504C4F54                   DEFM  'PLOTTER ERROR'
015F 54455220
0163 4552524F
0167 52
```

```
01 0168 0D                          DEFB  0DH
02 0169                     ;
03 0169                     ; PRINT ASCII
04 0169                     ;
05 0169                     CTYPE:  ENT
06 0169 01                          DEFB  1                 ;NO OF PARAMETER
07 016A 80                          DEFB  80H               ;CHR
08 016B                     TP:     DEFS  +2
09 016D                     ;
10 016D 2A6B01                      LD    HL,(TP)
11 0170 CD0000      E               CALL  .MOVE'
12 0173 EB                          EX    DE,HL
13 0174 3E50                        LD    A,50H             ;'P' PRINT CMD
14 0176 18CA                        JR    COUT
15 0178                     ;
16 0178                     ; SIZE ALPHA
17 0178                     ;
18 0178                     SBUF:   DEFS  +5
19 017D                     ;
20 017D                     SIZE:   ENT
21 017D 01                          DEFB  1                 ;NO OF PARAMETER
22 017E 00                          DEFB  0                 ;REAL
23 017F                     SE:     DEFS  +2
24 0181                     ;
25 0181 2A7F01                      LD    HL,(SE)
26 0184 117801                      LD    DE,SBUF
27 0187 011000                      LD    BC,0010H          ;15
28 018A CD3C02                      CALL  BTOA
29 018D EB                          EX    DE,HL
30 018E 3E53                        LD    A,53H             ;'S' SCALE CMD
31 0190 18B0                        JR    COUT
32 0192                     ;
33 0192                     ; DOT LINE
34 0192                     ;
35 0192                     DBUF:   DEFS  +5
36 0197                     ;
37 0197                     DOT:    ENT
38 0197 01                          DEFB  1                 ;NO OF PARAMETER
39 0198 00                          DEFB  0                 ;REAL
40 0199                     DT:     DEFS  +2
41 019B                     ;
42 019B 2A9901                      LD    HL,(DT)
43 019E 119201                      LD    DE,DBUF
44 01A1 018000                      LD    BC,0080H
45 01A4 CD3C02                      CALL  BTOA
46 01A7 EB                          EX    DE,HL
47 01A8 3E42                DOTWR:  LD    A,42H             ;'B' LINE GAGE CMD
48 01AA CD4201                      CALL  COUT
49 01AD 218C02                      LD    HL,DOTMSG
50 01B0 1890                        JR    COUT
```

```
01 01B2                 ;
02 01B2                 DOTM:   ENT
03 01B2 00                      DEFB  0                 ;NO PARAMETER
04 01B3 218902                  LD    HL,DOIMSG
05 01B6 18F0                    JR    DOTWR
06 01B8                 LINE:   ENT
07 01B8 00                      DEFB  0                 ;NO PARAMETER
08 01B9 218602                  LD    HL,LTMSG
09 01BC 180A                    JR    CNCT
10 01BE                 CSIZE:  ENT
11 01BE 00                      DEFB  0                 ;NO PARAMETER
12 01BF 218F02                  LD    HL,ISMSG
13 01C2 1804                    JR    CNCT
14 01C4                 ;
15 01C4                 ; PEN UP
16 01C4                 ;
17 01C4                 PUP:    ENT
18 01C4 00                      DEFB  0                 ;NO PARAMETER
19 01C5 219202                  LD    HL,PUMSG
20 01C8 C34501          CNCT:   JP    PLOT1
21 01CB                 ;
22 01CB                 ; PEN DOWN
23 01CB                 ;
24 01CB                 PDOWN:  ENT
25 01CB 00                      DEFB  0                 ;NO PARAMETER
26 01CC 219702                  LD    HL,PDMSG
27 01CF 18F7                    JR    CNCT
28 01D1                 ;
29 01D1                 ; ERROR CLEAR
30 01D1                 ;
31 01D1                 ERCLR:  ENT
32 01D1 00                      DEFB  0                 ;NO PARAMETER
33 01D2 1806                    JR    LFOUT
34 01D4                 ;
35 01D4                 ; HOME
36 01D4                 ;
37 01D4                 HOME:   ENT
38 01D4 00                      DEFB  0                 ;NO PARAMETER
39 01D5 3E48                    LD    A,48H             ;'H' HOME
40 01D7 CD2C02                  CALL  DOUT
41 01DA 3E0A            LFOUT:  LD    A,0AH             ;ER CLEAR CMD
42 01DC 184E                    JR    DOUT
43 01DE                 ;
44 01DE                 ; ROTATE ALPHA
45 01DE                 ;
46 01DE                 PBUF:   DEFS  +5
47 01E3                 ;
48 01E3                 ANGL:   ENT
49 01E3 01                      DEFB  1                 ;NO OF PARAMETER
50 01E4 00                      DEFB  0                 ;REAL
```

```
01 01E5                    AL:     DEFS  2
02 01E7                    ;
03 01E7 2AE501                     LD    HL,(AL)
04 01EA 11DE01                     LD    DE,PBUF
05 01ED 010400                     LD    BC,0004H
06 01F0 CD3C02                     CALL  BTOA
07 01F3 EB                         EX    DE,HL
08 01F4 3E51                       LD    A,51H          ;'Q' ROTATE CMD
09 01F6 C34201                     JP    COUT
10 01F9                    ;
11 01F9                    ; MARK
12 01F9                    ;
13 01F9                    MBUF:   DEFS  +5
14 01FE                    ;
15 01FE                    MARK:   ENT
16 01FE 01                         DEFB  1              ;NO OF PARAMETER
17 01FF 00                         DEFB  0              ;REAL
18 0200                    MK:     DEFS  +2
19 0202                    ;
20 0202 2A0002                     LD    HL,(MK)
21 0205 CD0000     E               CALL  ..INT0
22 0208 DA0000     E               JP    C,ER3
23 020B CD9C02                     CALL  CKNEG
24 020E 010700                     LD    BC,0007H
25 0211 CD7702                     CALL  CKCOD0
26 0214 11F901                     LD    DE,MBUF
27 0217 CD0000     E               CALL  CASC'
28 021A EB                         EX    DE,HL
29 021B 3E4E                       LD    A,4EH          ;'N' MARK CND
30 021D C34201                     JP    COUT
31 0220                    ;
32 0220                    ; AXIS SUB
33 0220                    ;
34 0220 F5                 AXIS:   PUSH  AF
35 0221 3E58                       LD    A,58H          ;'X' AXIS CMD
36 0223 CD2C02                     CALL  DOUT
37 0226 F1                         POP   AF
38 0227 CD2C02                     CALL  DOUT
39 022A 3E2C               AXIS1:  LD    A,2CH          ;',' DELIMITER
40 022C                    ;
41 022C                    ;
42 022C                    ;;;;;;;;;;
43 022C                    ;
44 022C                    ; PLOTTER DATA CONTROL
45 022C                    ;         A=DATA
46 022C                    ;
47 022C                    ;;;;;;;;;;
48 022C                    ;
49 022C F5                 DOUT:   PUSH  AF
50 022D DB0E               DOUTP:  IN    A,(PLTS)
```

```
01 022F CB47                        BIT   0,A
02 0231 20FA                        JR    NZ,DOUTP
03 0233 AF                          XOR   A
04 0234 D30F                        OUT   (PLTD),A
05 0236 F1                          POP   AF
06 0237 F680                        OR    80H
07 0239 D30F                        OUT   (PLTD),A
08 023B C9                          RET'
09 023C                     ;
10 023C                     ; BINARY TO ASCII
11 023C                     ;
12 023C CD0000      E       BTOA:   CALL  ..INT0
13 023F DA0000      E               JP    C,ER3
14 0242 CD9C02                      CALL  CKNEG
15 0245 CD7C02                      CALL  CKCOD
16 0248 CD0000      E               CALL  CASC'
17 024B C9                          RET
18 024C                     ;
19 024C                     ;
20 024C CD0000      E       NSBTOA: CALL  ..INT0
21 024F DA0000      E               JP    C,ER3
22 0252 CD5C02                      CALL  INS2D
23 0255 CD7C02                      CALL  CKCOD
24 0258 CD0000      E               CALL  CASC'
25 025B C9                          RET
26 025C                     ;
27 025C                     ; IF NEG... INSERT '-'
28 025C                     ;
29 025C 7C                  INS2D:  LD    A,H
30 025D CB7F                        BIT   7,A
31 025F C8                          RET   Z
32 0260 3E2D                        LD    A,2DH             ;'-'
33 0262 12                          LD    (DE),A
34 0263 13                          INC   DE
35 0264 7D                          LD    A,L
36 0265 2F                          CPL
37 0266 6F                          LD    L,A
38 0267 7C                          LD    A,H
39 0268 2F                          CPL
40 0269 67                          LD    H,A
41 026A 23                          INC   HL
42 026B C9                          RET
43 026C                     ;
44 026C                     ; SEEK 0D,INSERT 2C
45 026C                     ;
46 026C 13                          INC   DE
47 026D 1A                  INS2C:  LD    A,(DE)
48 026E FE0D                        CP    0DH
49 0270 20FA                        JR    NZ,INS2C-1
50 0272 3E2C                        LD    A,2CH
```

```
01 0274 12                      LD    (DE),A
02 0275 13                      INC   DE
03 0276 C9                      RET
04 0277                 ;
05 0277                 ; CHECK DATA VALUE
06 0277                 ;
07 0277 7C              CKCOD0: LD    A,H
08 0278 B5                      OR    L
09 0279 CA0000      E           JP    Z,ER3
10 027C E5              CKCOD:  PUSH  HL
11 027D B7                      OR    A
12 027E ED42            CKMAX:  SBC   HL,BC
13 0280 F20000      E           JP    P,ER3
14 0283 E1                      POP   HL
15 0284 B7                      OR    A
16 0285 C9                      RET
17 0286                 ;
18 0286                 ; LINE MSG
19 0286                 ;
20 0286 4C              LTMSG:  DEFB  4CH             ;'L' LINE CMD
21 0287 30                      DEFB  30H             ;'0'
22 0288 0D                      DEFB  0DH
23 0289                 ;
24 0289                 ; BETWEEN DOT
25 0289                 ;
26 0289 33              DOIMSG: DEFB  33H             ;'3'
27 028A 30                      DEFB  30H             ;'0'
28 028B 0D                      DEFB  0DH
29 028C                 ;
30 028C                 ; DOT MSG
31 028C                 ;
32 028C 4C              DOTMSG: DEFB  4CH             ;'L' LINE CMD
33 028D 31                      DEFB  31H             ;'1'
34 028E 0D                      DEFB  0DH
35 028F                 ;
36 028F                 ; INITIAL SIZE SET
37 028F                 ;
38 028F 53              ISMSG:  DEFB  53H             ;'S'
39 0290 34                      DEFB  34H             ;'4'
40 0291 0D                      DEFB  0DH
41 0292                 ;
42 0292                 ; PEN UP MSG
43 0292                 ;
44 0292 52              PUMSG:  DEFB  52H             ;'R' RELATIVE CMD
45 0293 30                      DEFB  30H             ;'0'
46 0294 2C                      DEFB  2CH             ;','
47 0295 30                      DEFB  30H             ;'0'
48 0296 0D                      DEFB  0DH
49 0297                 ;
50 0297                 ; PEN DOWN MSG
```

```
01 0297                    ;
02 0297 49                 PDMSG:  DEFB  49H               ;'I' INCREASE CMD
03 0298 30                         DEFB  30H               ;'0'
04 0299 2C                         DEFB  2CH               ;','
05 029A 30                         DEFB  30H               ;'0'
06 029B 0D                         DEFB  0DH
07 029C                    ;
08 029C                    ; CHECK DATA SIGN
09 029C                    ;
10 029C 7C                 CKNEG:  LD    A,H
11 029D CB7F                       BIT   7,A
12 029F C8                         RET   Z
13 02A0 CD0000      E      CKER2:  CALL  BEERR
14 02A3 51                         DEFB  51H               ;81
15 02A4 554E464F                   DEFM  'UNFORMAT DATA ERROR'
16 02A8 524D4154
17 02AC 20444154
18 02B0 41204552
19 02B4 524F52
20 02B7 0D                         DEFB  0DH
21 02B8                    ;
22 02B8                            END
```

AL	01E5	ANGL	01E3	AXIS	0220	AXIS1	022A	AXOUT	00D2
BTOA	023C	CKCOD	027C	CKCOD0	0277	CKER	0152	CKER1	0157
CKER2	02A0	CKMAX	027E	CKNEG	029C	CNCT	01C8	COUT	0142
CSIZE	01BE	CTYPE	0169	DBUF	0192	DOIMSG	0289	DOT	0197
DOTM	01B2	DOTMSG	028C	DOTWR	01A8	DOUT	022C	DOUTP	022D
DT	0199	ERCLR	01D1	HOME	01D4	IMBUFF	0036	IMOVE	0046
IMX	0048	IMY	004B	INS2C	026D	INS2D	025C	IPBUFF	00D8
IPLOT	00E8	IPX	00EA	IPY	00ED	ISMSG	028F	LFOUT	01DA
LINE	01B8	LTMSG	0286	MARK	01FE	MBUF	01F9	MBUFF	0000
MK	0200	MOVE	0010	MX	0012	MY	0015	NSBTOA	024C
PBUF	01DE	PDMSG	0297	PDOWN	01CB	PLOT	011D	PLOT1	0145
PLTD	000F	PLTS	000E	PTBUFF	010D	PUMSG	0292	PUP	01C4
PX	011F	PY	0122	QRXBUF	006C	QRYBUF	00A0	QX	007E
QY	00B2	RX	0081	RY	00B5	SBUF	0178	SE	017F
SIZE	017D	TP	016B	XAXIS	007C	YAXIS	00B0		

—— BASIC MAIN PROGRAM ——

```
     BASIC COMPILER SP-7715   <X-VDEM> PAGE 1          12/23/80

000F  EXTERNAL PLOT,DOTM,IPLOT,MOVE,IMOVE,PUP,PDOWN,DOT,XAXIS,YAXIS,CTYPE
000F  EXTERNAL CSIZE,SIZE,LINE,HOME,ANGL,MARK,ERCLR
000F  10 DIM D(1,255),E(50)
004C  15 DIM A(23),B(23)
0073  20 HOME : CSIZE : ANGL 0: LINE
00B4  100 REM  *FDOS NO SAKUGA*
00BA  105 REM  * F *
00C0  110 MOVE 200,2400
00E9  120 IPLOT 400,0: IPLOT 0,-150: IPLOT -400,0: IPLOT 0,150
0172  130 IMOVE 50,-50: IPLOT 300,0: IPLOT 0,-50: IPLOT -300,0: IPLOT 0,50
0203  140 MOVE 200,2170: IPLOT 400,0: IPLOT 0,-150: IPLOT -230,0
027D  145 IPLOT 0,-220: IPLOT -170,0: IPLOT 0,370
02E9  150 IMOVE 50,-50: IPLOT 300,0: IPLOT 0,-50
0340  155 IPLOT -230,0: IPLOT 0,-220: IPLOT -70,0
03A7  160 IPLOT 0,270
03C9  165 REM  * D *
03CF  170 MOVE 630,2400: IPLOT 210,0
040D  180 A=840:B=300:C=2100: GOSUB 340
0454  190 IPLOT -210,0: IPLOT 0,370: IPLOT 170,0: IPLOT 0,-220
04C0  200 A=810:B=150:C=2100: GOSUB 280
04FB  210 IPLOT -180,0: IPLOT 0,150: IMOVE 50,-50: IPLOT 160,0
0575  220 A=840:B=250:C=2100: GOSUB 340
05B0  230 IPLOT -160,0: IPLOT 0,270: IPLOT 70,0: IPLOT 0,-220: IPLOT 60,0
0638  240 A=810:B=200:C=2100: GOSUB 280
066C  250 IPLOT -130,0: IPLOT 0,50: GOTO 400
06B7  280 FOR I=-π/2 TO π/2 STEP π/60
0711  290 X=INT(A+B*COS(I)):Y=INT(C+B*SIN(I))
07AD  300 PLOT X,Y
07C8  310 NEXT I
07D9  320 RETURN
07E5  340 FOR I=π/2 TO -π/2 STEP -π/60
082D  350 X=INT(A+B*COS(I)):Y=INT(C+B*SIN(I))
08B5  360 PLOT X,Y
08D0  370 NEXT I
08DC  390 RETURN
08E8  395 REM  * O *
08EE  400 X=0:Y=0:A=0:B=0:C=0:D=0: MOVE 1760,2100
0952  420 A=1460:B=300:C=2100:D=300: GOSUB 540
0998  430 IMOVE -50,0
09BC  440 A=1460:B=250:C=2100:D=250: GOSUB 540
09FB  450 IMOVE -90,0
0A26  460 A=1460:B=160:C=2100:D=200: GOSUB 540
0A65  470 IMOVE -50,0
0A89  480 A=1460:B=110:C=2100:D=150: GOSUB 540
0ACF  490 MOVE 1840,2350: GOTO 600
0AFE  540 FOR I=0*π TO 2*π STEP π/60
0B34  550 X=INT(A+B*COS(I)):Y=INT(C+D*SIN(I))
0BBC  560 PLOT X,Y
0BD7  570 NEXT I:A=0:B=0:C=0:D=0:X=0:Y=0: RETURN
0C2B  590 REM  * S *
0C31  600 FOR I=π TO π+7/6 STEP π/60
0C80  610 X=INT(1960+120*COS(I)):Y=INT(2350+120*SIN(I))
0D16  620 PLOT X,Y: NEXT I:X=0:Y=0
0D4D  630 A=1920:B=110:C=2010: GOSUB 830
0D8F  640 IPLOT -120,0: IPLOT 0,-50: IPLOT 140,0
```

```
0DED  650 A=1940:B=160:C=2010: GOSUB 880
0E28  660 FOR I=π*7/6 TO π*186/180 STEP -π/60
0E86  670 X=INT(2030+135*COS(I)):Y=INT(2370+135*SIN(I))
0F23  680 PLOT X,Y
0F3E  690 NEXT I: IPLOT -60,0:X=0:Y=0: IMOVE -38,45
0FAA  700 IPLOT 150,0
0FC5  710 FOR I=π*8/9 TO π*7/6 STEP π/60
1021  720 X=INT(2030+85*COS(I)):Y=INT(2370+85*SIN(I))
10B0  730 PLOT X,Y
10CB  740 NEXT I:X=0:Y=0
10ED  750 A=1940:B=210:C=2010: GOSUB 830
1121  760 IPLOT -190,0: IPLOT 0,150: IPLOT 170,0
1176  770 A=1920:B=60:C=2010: GOSUB 880
11AA  780 FOR I=π*7/6 TO π*8/9 STEP -π/60
1201  790 X=INT(1960+170*COS(I)):Y=INT(2350+170*SIN(I))
1289  800 PLOT X,Y
12A4  810 NEXT I:X=0:Y=0: GOTO 930
12CC  820 REM  * SUB A *
12D2  830 FOR I=π/6 TO -π/2 STEP -π/60
131A  840 X=INT(A+B*COS(I)):Y=INT(C+B*SIN(I))
13A2  850 PLOT X,Y
13BD  860 NEXT I:X=0:Y=0: RETURN
13E5  870 REM  * SUB B *
13EB  880 FOR I=-π/2 TO π/6 STEP π/60
142A  890 X=INT(A+B*COS(I)):Y=INT(C+B*SIN(I))
14B2  900 PLOT X,Y
14CD  910 NEXT I:X=0:Y=0: RETURN
14F5  920 REM  * DIAMOND *
14FB  925 MOVE 3000,2050
1524  930 FOR Z=1 TO 23
154C  940 A(Z)=INT(350*COS(2*π*(Z-1)/23+π/2)+3000)
15FE  950 B(Z)=INT(-350*SIN(2*π*(Z-1)/23+π/2)+2050)
16A3  960 NEXT Z
16AF  970 FOR S=2 TO 11
16DE  975 L=2-S
16F0  980 X=A(23):Y=B(23)
172B  982 MOVE X,Y
1746  985 FOR I=1 TO 24
1775  990 J=L+S
1787  995 IF J<24 THEN 1010
17AF 1000 J=J-23
17C1 1010 X1=A(J):L=J:Y1=B(J)
1802 1020 A=X1-X:B=Y1-Y
1820 1025 IPLOT A,B
183B 1030 X=X1:Y=Y1
1857 1040 NEXT I: NEXT S:X=0:Y=0:X1=0:Y1=0:A=0:B=0
18AB 1050 MOVE 1650,1320
18D4 1090 REM  *PRINT "  3D-GRAPH"*
18DA 1100 A$="  3D-GRAPH"
1904 1110 FOR I=3 TO 0 STEP -1
193C 1115 MOVE 1650,1320
1957 1120 ANGL I
196C 1130 CTYPE A$
1989 1140 NEXT I:A$=""
19A5 1190 REM  *  3D-GRAPHIC  *
```

```
19AB  1195 T=0:S=0
19C7  1200 FOR L=0 TO 255
19EF  1210 D(0,L)=-1:D(1,L)=-1: NEXT L
1A2D  1215 MOVE 100,250
1A4F  1220 FOR Y=-180 TO 180 STEP 4
1A87  1230 FOR X=-180 TO 180 STEP 4
1AB8  1240 IF (Y=-180)*(X=180) THEN 1800
1AFC  1250 R=π/180*SQR(X*X+Y*Y)
1B4D  1260 Z=100*COS(R)-30*COS(3*R)
1BA8  1270 A=INT(110+X/2+(16-Y/2)/2)
1C17  1280 B=INT((116-Y/2-Z)/2):B=192-B
1C81  1290 IF (A<0)+(A>255) THEN 1330
1CBC  1300 IF D(0,A)=-1 THEN 1370
1CF3  1310 IF B<=D(0,A) THEN 1440
1D26  1320 IF B>=D(1,A) THEN 1460
1D59  1330 NEXT X
1D65  1340 IMOVE 0,0:T=0:S=0
1D96  1350 NEXT Y
1DA2  1360 GOTO 2000
1DAE  1370 IF A=0 THEN 1430
1DCC  1380 IF D(0,A-1)=-1 THEN 1430
1E0F  1390 IF D(0,A+1)=-1 THEN 1430
1E52  1400 D(0,A)=INT((D(0,A-1)+D(0,A+1))/2)
1ED2  1410 D(1,A)=INT((D(1,A-1)+D(1,A+1))/2)
1F52  1420 GOSUB 1480: GOTO 1330
1F6B  1430 D(0,A)=B:D(1,A)=B: GOSUB 1480: GOTO 1330
1FBA  1440 GOSUB 1480:D(0,A)=B: IF D(1,A)=-1 THEN D(1,A)=B
2031  1450 GOTO 1330
203D  1460 GOSUB 1480:D(1,A)=B: IF D(0,A)=-1 THEN D(1,A)=B
20B4  1470 GOTO 1330
20C0  1475 REM  *SUB DATA OUT *
20C6  1480 DX=100+7*A:DY=8*B
20F0  1500 MOVE DX,DY: PDOWN : PUP
211D  1510 RETURN
2129  1790 REM  *SUB GENSUI *
212F  1800 FOR K=1 TO 50:E(K)=0: NEXT K
2175  1810 FOR I=1 TO 50
219D  1820 E(I)=SIN(I*π/12.5)
21E9  1830 NEXT I
21F5  1840 N=15
220D  1850 FOR I=1 TO N
2235  1860 D=5*I/N+5:G=(N-I+0.5)/N*4:O=DX:P=DY
22B3  1870 MOVE O,P
22CE  1880 FOR J=1 TO 50
22F6  1890 O=DX+E(J)*G*(50-J):P=P+D
2345  1900 PLOT O,P
2360  1910 NEXT J: NEXT I: MOVE DX,DY: GOTO 1250
238D  1990 REM  * SIN(X)/X *
2393  2000 MOVE 2350,900
23B5  2020 XAXIS 87,12
23DE  2030 MOVE 2870,700
2407  2040 YAXIS 200,4: MOVE 2350,900
2437  2050 FOR I=-6 TO 6 STEP 0.08
246F  2060 IF I=0 THEN Y=1:X=0: GOTO 2090
24A6  2070 X=I*π
```

```
24B8  2080 Y=SIN(X)/X
24D9  2090 A=2870+INT(27.5*X+0.5)
2519  2100 B=900+INT(400*Y)
2546  2110 PLOT A,B
2561  2120 NEXT I
256D  2130 MOVE 2350,1300: DOTM : MARK 1
25A7  2140 IPLOT 174,0: MARK 2: IPLOT 174,-200: MARK 3: IPLOT 174,-200: MARK 4
2632  2150 IPLOT 174,200: MARK 5: IPLOT 174,200: MARK 6: IPLOT 174,0
2695  2160 LINE
26A4  2170 MOVE 2350,700: IPLOT 0,800: IPLOT 1044,0: IPLOT 0,-800: IPLOT -1044,0
2733  2180 MOVE 3100,800: SIZE 2:A$="XAXIS": CTYPE A$
278B  2190 A$="YAXIS": MOVE 2800,1300: ANGL 1: CTYPE A$: ANGL 0:A$=""
2802  2390 REM  * SHARP SIZE *
2808  2400 MOVE 2550,300
282A  2410 B$="SHARP "
2846  2420 FOR I=1 TO 15 STEP 5
2879  2430 SIZE I
288E  2440 CTYPE B$
28A6  2450 NEXT I:B$=""
28C2  2460 CSIZE
28D1  2790 REM  * CHARACTOR SET *
28D7  2800 MOVE 400,150
28F2  2810 C$="1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ !#$%&()+-,./*<>@?:;"
2940  2820 CTYPE C$
2958  3000 REM  * LINE TYPE *
295E  3010 HOME : PLOT 0,2500: PLOT 3590,2500: PLOT 3590,0: PLOT 0,0: IMOVE 15,1
5
29E4  3020 H=3560:V=2470:R=15
2A19  3030 FOR I=20 TO 60 STEP 20
2A53  3040 DOT I
2A68  3050 IPLOT 0,V: IPLOT H,0: IPLOT 0,-V: IPLOT -H,0: IMOVE R,R
2AE9  3060 V=V-2*R:H=H-2*R
2B1F  3070 NEXT I
2B2B  5000 HOME : IMOVE 60,60
2B4F  5010 LINE : IPLOT 0,V: IPLOT H,0: IPLOT 0,-V: IPLOT -H,0
2BC4  5100 D$="PEN UP DOWN"
2BE5  5110 MOVE 2050,400
2C00  5120 CTYPE D$
2C18  5130 IMOVE 50,0
2C33  5140 FOR I=1 TO 5: PUP : PDOWN : NEXT I
2C73  CSIZE : LINE : ANGL 0: HOME
2C9D  19999 END
** COMPILER FOUND NO ERRORS.
```

Library/Package

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

——CONTENTS——

USING LIBRARY ROUTINES

The FDOS master diskette contains three libraries (MON. LIB, FDOS. LIB and RELO. LIB).

MON. LIB is a file of monitor subroutine addresses defined with the EQU statement. That is, it contains a program such as that shown at right which has been assembled and converted to the library mode.

```
GETL:   EQU 0003H
LETNL:  EQU 0006H
PRNTS:  EQU 000CH
PRNT:   EQU 0012H
MSG:    EQU 0015H
MSGX:   EQU 0018H
GETKY:  EQU 001BH
BRKEY:  EQU 001EH
MELDY:  EQU 0030H
          ⋮
```

1- Part of the contents of MON . ASC

Monitor subroutines are often used when creating programs with the assembler; in such cases, they are used as described below.

First, the subroutine names are written as is (without addresses defined) as external names when the program is written. These are then assembled with the assembler. When the assembly listing is reviewed at this time, the symbol "E" is affixed to indicate that the names are external names. Next, the program is linked; MON. LIB is linked at this time also. For example,

```
2 > LINK GAMEPRG1, $FD1 ;MON. LIB
       ↑
  Program created
```

MON . LIB must be written last at this time.

FDOS . LIB is a file of subroutine addresses in FDOS which are defined with the EQU statement; it is used in the same manner as MON . LIB. Since MON . LIB is contained in FDOS . LIB, it is only necessary to link FDOS . LIB when both monitor and FDOS subroutines are to be used at the same time.

RELO . LIB is a library of subroutines for programs created with the BASIC compiler. It contains subroutines for the four basic arithmetic operations, functional calculations, character string processing, error message display and many others. In other words, whereas MON . LIB and FDOS . LIB are simple collections of EQU statements, RELO . LIB contains actual subroutines.

When the relocating loader is used for linkage with RELO . LIB, it is possible to select only the routines required from the many available for linkage.

RELO . LIB is used in the same manner as MON . LIB and FDOS . LIB. Further, the contents of MON . LIB and FDOS . LIB are included in RELO . LIB.

Source programs MON . ASC and FDOS . ASC are also included on the master diskette along with MON . LIB and FDOS . LIB. It is possible to modify and add to the libraries by regenerating the source programs to recreate the libraries as necessary.

Note Detailed procedures for using FDOS . LIB are contained in "LINKING ASSEMBLY PROGRAM WITH FDOS" in Appendix; see "EXAMPLE OF PLOTTER CONTROL APPLICATION" in Programming Utility for details on RELO . LIB.

MONITOR SUBROUTINES (MON. LIB)

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL LETNL (0006H)	To change the line and set the cursor to the beginning of the next line.	All registers except AF
CALL NL (0009H)	Changes the line and sets the cursor to its beginning if the cursor is not already located at the beginning of a line.	All registers except AF
CALL PRNTS (000CH)	Displays one space only at the cursor position on the display screen.	All registers except AF
CALL PRNT (0012H)	Handles data in ACC as ASCII code and displays it on the screen, starting at the cursor position. However, a carriage return is performed for 0DH and the various cursor control operations are performed for 11H-16H when these are included.	All registers except AF
CALL MSG (0015H)	Displays a message, starting at the cursor position on the screen. The starting address of the message must be specified in the DE register in advance. The message is written in ASCII code and must end in 0DH. A carriage return is not executed, however, cursor control codes (11H-16H) are.	All registers
CALL MSGX (0018H)	Almost the same as MSG, except that cursor control codes are for reverse character display.	All registers
CALL BELL (003EH)	Sounds a tone (approximately 880 Hz) momentarily.	All registers except AF
CALL MELDY (0030H)	Plays musical data. The starting address of the musical must be specified in advance in the DE register. As with BASIC, the musical interval and the duration of notes of the musical data are expressed in that order in ASCII code. The end mark must be either 0DH or C8H. The melody is over if CF is 0 when a return is made; if CF is 1 it indicates that [SHIFT] + [BREAK] were pressed.	All registers except AF
CALL XTEMP (0041H)	Sets the musical tempo. The tempo data (01 to 07) is set in and called from ACC. ACC ←01 Slowest ACC ←04 Medium speed ACC ←07 Fastest Care must be taken here to ensure that the tempo data is entered in ACC in binary code, and not in the ASCII code corresponding to the numbers 1 to 7 (31H to 37H).	All registers
CALL MSTA (0044H)	Continuously sounds a note according to a specified division factor. The division factor nn' consists of two bytes of data; n' is stored at address 11A1H and n is stored at address 11A2H. The relationship between the division factor and the frequency produced is 1 MHz/nn'.	BC and DE only

<table>
<tr><th>Subroutine name (hexadecimal address)</th><th>Function</th><th>Registers preserved</th></tr>
<tr><td>CALL MSTP
(0047H)</td><td>Discontinues a tone being sounded.</td><td>All registers except AF</td></tr>
<tr><td>CALL TIMST
(0033H)</td><td>Sets the built-in clock. (The clock is activated by this call.) The call conditions are:
ACC ← 0 (AM), ACC ← 1 (PM)
DE ← the time in seconds (2 bytes)</td><td>All registers except AF</td></tr>
<tr><td>CALL TIMRD
(003BH)</td><td>Reads the value of the built-in clock. The conditions upon return are:
ACC ← 0 (AM), ACC ← 1 (PM)
DE ← the time in seconds (2 bytes)</td><td>All registers except AF and DE</td></tr>
<tr><td>CALL BRKEY
(001EH)</td><td>Checks whether [SHIFT] + [BREAK] were pressed. ZF is set if they were pressed, and ZF is reset if they were not.</td><td>All registers except AF</td></tr>
<tr><td>CALL GETL
(0003H)</td><td>Inputs one line entered from the keyboard. The starting address where the data input is to be stored must be specified in advance in the DE register. [CR] functions as the end mark. 80 is the maximum number of characters which can be input (including the end mark 0DH).
Key input is displayed on the screen and cursor control is also accepted.
The BREAK code (64H) followed by a carriage return code (0DH) is set at the beginning of the address specified in the DE register when [SHIFT] + [BREAK] are pressed.</td><td>All registers</td></tr>
<tr><td>CALL GETKY
(001BH)</td><td>Takes one character only into ACC from the keyboard in ASCII code. A return is made after 00 is set in ACC if no key is pressed when the subroutine is executed. However, there is no protection against chattering and key input is not displayed on the screen.
Codes which are taken into ACC when these special keys are pressed are shown below. Here, "display code" refers to a code number which is used to call out a character from within the character generator.</td><td>All registers except AF</td></tr>
<tr><td>Special keys taken in by GETKY</td><td colspan="2"><table>
<tr><th>Special key</th><th>Display code</th><th>Code taken into ACC
SP-1002</th></tr>
<tr><td>DEL</td><td>C7</td><td>60</td></tr>
<tr><td>INST</td><td>C8</td><td>61</td></tr>
<tr><td>CAP</td><td>C9</td><td>62</td></tr>
<tr><td>SML</td><td>CA</td><td>63</td></tr>
<tr><td>BREAK</td><td>CB</td><td>64</td></tr>
<tr><td>CR</td><td>CD</td><td>66</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Subroutine name
(hexadecimal address)</th><th>Function</th><th>Registers
preserved</th></tr>
<tr><td>CALL PRTHL
(03BAH)</td><td>Displays the contents of the HL register on the display screen as a 4-digit hexadecimal number.</td><td>All registers
except AF</td></tr>
<tr><td>CALL PRTHX
(03C3H)</td><td>Displays the contents of the A register on the display screen as a 2-digit hexadecimal number.</td><td>All registers
except AF</td></tr>
<tr><td>CALL ASC
(03DAH)</td><td>Converts the contents of the lower 4 bits of ACC from hexadecimal to ASCII code and returns after setting the converted data in ACC.</td><td>All registers
except AF</td></tr>
<tr><td>CALL HEX
(03F9H)</td><td>Converts the 8 bits of ACC from ASCII code to hexadecimal and returns after setting the converted data in the lower 4 bits of ACC.
When CF=0 upon return ACC ← hexadecimal
When CF=1 upon return ACC is not assured</td><td>All registers
except AF</td></tr>
<tr><td>CALL HLHEX
(0410H)</td><td>Handles a consecutive string of 4 characters in ASCII code as hexadecimal string data and returns after setting the data in the HL register. The call and return conditions are as follows.
DE ← starting address of the ASCII string (string "3" "1" "A" "5")
↑—DE
CALL HLHEX
CF=0 HL ← hexadecimal number (e.g., HL = 31A5H)
CF=1 HL is not assured.</td><td>All registers
except AF
and HL</td></tr>
<tr><td>CALL 2HEX
(041FH)</td><td>Handles 2 consecutive ASCII strings as hexadecimal strings and returns after setting the data in ACC. The call and return conditions are as follows.
DE ← starting address of the ASCII string (e.g., "3" "A")
↑—DE
CALL 2HEX
CF=0 ACC ← hexadecimal number (e.g., ACC=3AH)
CF=1 ACC is not assured.</td><td>All registers
except AF
and DE</td></tr>
<tr><td>CALL ??KEY
(09B3H)</td><td>Awaits key input while causing the cursor to flash. When a key entry is made it is converted to display code and set in ACC, then a return is made.</td><td>All registers
except AF</td></tr>
<tr><td>CALL ?ADCN
(0BB9H)</td><td>Converts an ASCII value to display code. Call and return conditions are as follows.
ACC ← ASCII value
CALL ?ADCN
ACC ← display code</td><td>All registers
except AF</td></tr>
<tr><td>CALL ?DACN
(0BCEH)</td><td>Converts a display code to an ASCII value. Call and return conditions are as follows.
ACC ← display code
CALL ?DACN
ACC ← ASCII value</td><td>All registers
except AF</td></tr>
<tr><td>CALL ?BLNK
(0DA6H)</td><td>Checks vertical blanking of the display screen. Waits until the vertical blanking interval starts and then returns when blanking takes place.</td><td>All registers</td></tr>
<tr><td>CALL ?DPCT
(0DDCH)</td><td>Controls the display on the display screen. The relationship between ACC at the time of the call and control is as follows.
<table>
<tr><th>ACC</th><th>Control contents</th><th>ACC</th><th>Control contents</th></tr>
<tr><td>C0H</td><td>Scrolling</td><td>C6H</td><td>Same function as the [CLR] key</td></tr>
<tr><td>C1H</td><td>Same function as the ↓ key</td><td>C7H</td><td>Same function as the [DEL] key</td></tr>
<tr><td>C2H</td><td>Same function as the ↑ key</td><td>C8H</td><td>Same function as the [INST] key</td></tr>
<tr><td>C3H</td><td>Same function as the → key</td><td>C9H</td><td>Same function as the [CAP] key</td></tr>
<tr><td>C4H</td><td>Same function as the ← key</td><td>CAH</td><td>Same function as the [SML] key</td></tr>
<tr><td>C5H</td><td>Same function as the [HOME] key</td><td>CDH</td><td>Same function as the [CR] key</td></tr>
</table></td><td>All registers</td></tr>
</table>

Subroutine name (hexadecimal address)	Function	Registers preserved

CALL ?PONT (0FB1H)	Sets the current position of the cursor on the display screen in register HL. The return conditions are as follows.
	CALL ?PONT
	HL ⟵ cursor position on the display screen (V-RAM address)
	(Note) The X-Y coordinates of the cursor are contained in DSPXY (1171 H). The current position of the cursor is loaded as follows.
	LD HL, (DSPXY) ; H ⟵ Y coordinate on the screen
	L ⟵ X coordinate on the screen
	The cursor position is set as follows.
	LD (DSPXY), HL
		All registers except AF and HL

CALL WRINF (0021H)	Writes the current contents of a certain part of the header buffer (described later) onto the tape, starting at the current tape position. Return conditions CF=0 No error occurred. CF=1 The [BREAK] key was pressed.	All registers except AF
CALL WRDAT (0024H)	Writes the contents of the specified memory area onto the tape as a CMT data block in accordance with the contents of a certain part of the header buffer. Return conditions CF=0 No error occurred. CF=1 The [BREAK] key was pressed.	All registers except AF
CALL RDINF (0027H)	Reads the first CMT header found starting at the current tape position into a certain part of the header buffer. Return conditions CF=0 No error occurred. CF=1, ACC=1 A check sum error occurred. CF=1, ACC=2 The [BREAK] key was pressed.	All registers except AF
CALL RDDAT (002AH)	Reads in the CMT data block according to the current contents of a certain part of the header buffer. Return conditions CF=0 No error occurred. CF=1, ACC=1 A check sum error occurred. CF=1, ACC=2 The [BREAK] key was pressed.	All registers except AF
CALL VERFY (002DH)	Compares the first CMT file found following the current tape position with the contents of the memory area indicated by its header. Return conditions CF=0 No error occurred. CF=1, ACC=1 A match was not obtained. CF=1, ACC=2 The [BREAK] key was pressed.	All registers except AF

(Note) The contents of the header buffer at the specific addresses are as follows. The buffer starts at address 10F0H and consists of 128 bytes.

Address	Contents
IBUFE (10F0H)	This byte indicates one of the following file modes. 1. Object file (machine language program) 2. BASIC text file 3. BASIC data file 4. Source file (ASCII file) 5. Relocatable file (relocatable binary file) A0. PASCAL interpreter text file A1. PASCAL interpreter data file
IBU1 (10F1H ~ 1101H)	These 17 bytes indicate the file name . However, since 0DH is used as the end mark, in actuality the file name is limited to 16 bytes. Example: [S][A][M][P][L][E][0D]
IBU18 (1102H ~ 1103H)	These two bytes indicate the byte size of the data block which is to follow.
IBU20 (1104H ~ 1105H)	These two bytes indicate the data address of the data block which is to follow. The loading address of the data block which is to follow is indicated by "CALL RDDAT." The starting address of the memory area which is to be output as the data block is indicated by "CALL WRDAT".
IBU22 (1106H ~ 1107H)	These two bytes indicate the execution address of the data block which is to follow.
IBU24 (1108H ~)	These bytes are used for supplemental information, such as comments.

Example

Address	Content	
10F0	01	; indicates an object file (machine language program).
10F1	'S'	; the file name is "SAMPLE".
10F2	'A'	
10F3	'M'	
10F4	'P'	
10F5	'L'	
10F6	'E'	
10F7	0D	
10F8 ~ 1101	Variable	
1102	00	; the size of the file is 2000H bytes.
1103	20	
1104	00	; the data address of the file is 1200H.
1105	12	
1106	60	; the execution address of the file is 1260H.
1107	12	

FDOS SUBROUTINES (FDOS. LIB)

—— Outline ——

FDOS subroutines can be broadly divided into three groups.
That is,

1. CLI (Command Line Interpreter) subroutines
2. IOCS (Input Output Control System) subroutines
3. Utility subroutines

CLI subroutines are used to translate command lines appearing within user programs. That is, when programs are called in which switches and arguments appear in appended format (such as RUN PROG/P FILE1, FILE2 [CR]), these subroutines translate those switches and arguments.

IOCS subroutines are used to open and close files and devices. Utility subroutines are other general purpose subroutines.

Command lines are strings of characters (which have been converted to intermediate code) which are input from the keyboard as FDOS commands or other character strings in the same format. In the explanation below, except where otherwise indicated, command lines appear in intermediate code. See the table on page 30 for the intermediate codes.

—— CLI (Command Line Interpreter) ——

TRS10

Function: Converts FDOS command lines written in ASCII code into intermediate code.

Input registers: The HL register contains the starting address of the command line written in ASCII code. The DE register contains the starting address of the area storing the command line converted to intermediate code.

Calling procedure: CALL TRS10

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← error code)

Note: See the "System Error Messages" in System Command for details. The same applies below.

Registers preserved: All registers except AF.

- **CLI (Command Line Interpreter)**

Function: Interprets and executes FDOS command lines.

Input registers: The HL register contains the command line pointer.

Calling procedure:

```
LD    DE,(RJOB)
PUSH  DE
CALL  .CLI
POP   HL
LD    (RJOB),HL
```

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← FFH)

Example of use (DATE/P)

```
        LD    HL,DATE
        LD    DE,(RJOB)
        PUSH  DE
        CALL  .CLI
        POP   HL
        LD    (RJOB),HL
        JP    C,ERROR
        :
DATE:   DEFB  B1H  } Intermediate code
        DEFB  88H  } for DATE/P
        DEFB  0DH
```

Registers preserved: None

Caution: The LIMIT, RUN, EXEC and DEBUG commands cannot be executed. See page 29 for the RJOB.

?HEX (Check Hexadecimal)

Function: Converts a four digit, hexadecimal data item starting with "$" into sixteen bit, binary notation.

Input registers: HL contains the pointer; it should specify "$."

Calling procedure: CALL ?HEX

Output registers:
- CF=1 ..... Not a hexadecimal number. (ACC ← 3, and HL are preserved)
- CF=0 ..... a hexadecimal number. (DE ← data, HL indicates the address following the hexadecimal number)

Registers preserved: All registers except AF, DE and HL.

?SEP (Check Separator)

Function: Checks whether the contents of the address indicated by the HL register are a separator (one of the following: [CR] ␣ , : /).

Input registers: Register HL is the pointer.

Calling procedure: CALL ?SEP

Output registers:
- CF=1 ..... Not a separator. ACC ← 3 (error code) and the HL register are preserved.
- CF=0 ..... A separator.
  - ACC=2CH .... The separator is a space or a comma (the HL register then points to the address following the separator)
  - ACC=0DH .... The separator is [CR] or " / " (the HL register points to the separator)
  - ACC=3AH..... The separater is a colon " : " (the HL register points to the separator)

Registers preserved: All registers except AF and HL.

?GSW (Check Global Switch)

Function: Determines whether the global switch on the command line is correct and, if so, stores it in the area within FDOS.

Input registers: The DE register contains the starting address of the switch table. The HL register contains the command line pointer which points to the global switch.

Calling procedure:

```
        LD    DE,SWTBL
        CALL  ?GSW
        :
SWTBL : DEFB  SW1     ┐
        DEFB  SW2     │ List of items which may be used as global switches
        :             │ (these are written in intermediate code, from 0 to a
        DEFB  SWn     ┘ maximum of 5. See page 30)
        DEFB  FFH       End of table
```

Output registers:
CF=1 ..... Error (ACC ← error code)
CF=0 ..... Normal. The HL register points to the address following the global switch.

Registers preserved: All registers except AF, DE and HL.

TESW (Test Global Switch)

Function: Determines the presence or absence of the specified global switch. Subroutine "?GSW" must be called before this subroutine is used.

Input registers: None

Calling procedure:
CALL TESW
DEFB global switch

Output registers:
CF=0
... The specified global switch is present.
CF=1
... The specified global switch is not present.

Registers preserved: All registers except AF.

Example: This routine outputs whether or not global switch /P is present to the line printer or the CRT.

```
CALL  TESW
DEFB  88H          ; intermediate code for /P
PUSH  AF
CALL  C,MSG        ; displayed on the CRT if the switch
                   ;  is not present.
POP   AF
CCF                ; CF←CF (complement)
CALL  C,PMSG       ; printed on the line printer if the
                   ;  switch is present.
JP    C,ERROR      ; indicates a line printer error.
```

?LSW (Check Local Switch)

Function: Used to determine the local switch which is attached to the file name on the command line.

Input registers: The HL register is the command line pointer which indicates the start of the file name.

Calling procedure: CALL ?LSW

Output registers:
CF=1 ..... Error (ACC ← error code)
CF=0 ..... Normal
ZF=1 ..... No local switch. (ACC ← 0)
ZF=0 ..... Local switch is present.
(ACC ← intermediate code for the local switch)

Registers preserved: All registers except AF.

Example: Read-opens (ROPEN) a file with logical number 2 if a local switch is not present; if local switch /O is present the file is write-opened (WOPEN) with logical number 3; otherwise, an error occurs.

```
        EXX
        LD      B,4             ;dafault file mode .ASC
        EXX
        CALL    ?LSW
        JP      C,ERROR
        JR      NZ,L2
        LD      C,2             ;logical number 2
        CALL    ROPEN
        JR      L3
L2:     CP      89H             ;intermediate code for /0
        LD      A,8             ;error code (IL LOCAL SWITCH)
        JP      NZ,ERROR
        LD      C,3             ;logical number 3
        CALL    WOPEN
L3:     JP      C,ERROR
```

ROPEN (Read Open)

Function: Read-opens a file (including the input/output device).

Input registers: HL: Pointer which indicates the start of the file name.
C : Logical number (See note 3)
B' : Default file mode (See note 1)

Calling procedure: CALL ROPEN

Output registers: CF=1 ..... Error (ACC ← error code)
CF=0 ..... Normal
HL: Pointer (indicates the next separator)
B' : File mode (See note 1)
C' : File attribute (See note 2)
L' : Device number
IY : Starting address of the device table (See note 4)

Registers preserved: Only registers BC, DE and IX.

Example (when $FD1 ;ABC)

```
        LD      HL,FL
        LD      C,2 (logical number)
        EXX
        LD      B,4       (.ASC)
        EXX
        CALL    ROPEN
        CALL    C,ERR (See page 26)
        RET     C
          :
FL :DEFB  90H   ($FD1)
    DEFM  ';ABC'
    DEFB  0DH
```

WOPEN (Write Open)

Function: Write-opens a file (including an input/output device).

Input registers: HL: Pointer which indicates the start of the file name.
C : Logical number (See note 3)
B' : Default file mode (See note 1)

Calling procedure: CALL WOPEN

Output registers: CF=1 ..... Error (ACC ← error code)
CF=0 ..... Normal
HL: Pointer (Indicates the next separator)
B' : File mode (See note 1)
C' : File attribute (only for "0")
L' : Device number
IY : Starting address of the device table (See note 4)

Registers preserved: Only registers BC, DE and IX.

Example ($PTP/PE/LF)

```
        LD      HL,PTP
        LD      C,3 (logical number)
        EXX
        LD      B,4       (.ASC)
        EXX
        CALL    WOPEN
        JP      C,ERROR
          :
PTP :DEFB  A1H   ($PTP)
    DEFB  8FH   (/PE)
    DEFB  8CH   (/LF)
    DEFB  0DH
```

MODECK (Filemode Check)

Function: Checks whether the file mode indicated in register B' for the file opened is correct or not.

Input registers: Register B' contains the file mode of the opened file.

Calling procedure: CALL MODECK
DEFB file mode number (see page 30 concerning file modes)

Output registers: CF=0 ..... The file mode is correct.
CF=1 ..... The file mode is not correct. ACC ← error code

Registers preserved: All registers except AF.

(Note 1) The default file mode is the mode which is assumed when no mode is specified in the command line. The numbers enclosed in parentheses indicate the file mode number. (see page 30.)

Example

Command line	Default file mode	Actual file mode
ABC . ASC	. ASC (4)	. ASC (4)
ABC . LIB	. RB (5)	. LIB (7)
ABC	. OBJ (1)	. OBJ (1)
ABC	. ASC (4)	. ASC (4)

(Note 2) The file attribute indicates the protection of file access, and is expressed as one of the following ASCII codes.

"0" a file with no protection.

"R" a file for which reading is inhibited. (Read protected file)

"W" a file for which writing is inhibited. (Write protected file)

"P" a file for which both reading and writing are inhibited. (Permanent file). However, files with the attribute "P" can be executed if the file mode is OBJ. The EXEC command can be executed if the file mode is ASC.

Normally, the programmer does not need to be aware of file attributes since they are managed by FDOS.

(Note 3) Logical file numbers are numbers within FDOS which have a one-to-one correspondence wtih physical files opened (including input/output devices). Numbers from 1 to 249 may be used as logical numbers; however, since programs within FDOS use all of the numbers from 128 on, user programs should use only the numbers from 1 to 127 to avoid conflict.

(Note 4) An explanation of the device table is contained in "USER I/O ROUTINE" in Appendix; however, except for special I/O operations, the programmer normally does not need to be aware of the contents of the device table.

GET1L (Get 1 Line)

Function: Reads in one line from the file whose logical number is specified in the C register. The line read is one which is terminated with 0DH. The data read is stored in the area indicated by the address in the DE register. The length of the line, including 0DH, must be no more than 128 bytes.

Input registers: The C register contains the logical number. The DE register contains the address of the area in which the data is stored.

Calling procedure: CALL GET1L

Output registers: CF=0 ..... Normal
CF=1, ACC=0 ..... File end
CF=1, ACC≠0 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

GET1C(Get 1 Charcter)

Function: Reads one byte from the file whose logical number is specified in the C register.

Input registers: The C register contains the logical number.

Calling procedure: CALL GET1C

Output registers: CF=0 ..... Normal (ACC ← data read)
CF=1, ACC=0 ..... File end
CF=1, ACC≠0 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

GETBL (Get Block)

Function: Reads data into the address indicated in the DE register from the file whose logical number is specified in the C register; only the number of bytes of data indicated in the HL register are read in.

Input registers: The C register contains the logical number. The DE register contains the address in which the data is to be stored. The HL register contains the number of bytes of data to be read.

Calling procedure: CALL GETBL

Output registers: CF=0 ..... Normal ⎫ DE ← address of the next block of data to be read
CF=1, ACC=0 ..... File end ⎭ HL ← number of bytes of data actually read
CF=1, ACC≠0 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC and IX.

?EOF (Check End-of-file)

Function: Checks for the end of a read-opened file. ZF becomes "1" when an attempt is made to read beyond the end of data.

Input registers: The C register contains the logical number.

Calling procedure: CALL ?EOF

Output registers: CF=1 ..... Error (ACC ← error code)
CF=0, ZF=1 ..... Not file end
CF=0, ZF=0 ..... File end
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

PUT1C (Put 1 Character)

Function: Outputs one byte of data to the file whose logical number is specified in the C register.

Input registers: The C register contains the logical number. The ACC register contains the data to be output.

Calling procedure: CALL PUT1C

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

PUT1L (Put 1 Line)

Function: Outputs the line starting at the address specified in the DE register to the file whose logical number is specified in the C register. Outputs the ending carriage return.

Input registers: The C register contains the logical number. The DE register contains the starting address of the data to be output.

Calling procedure: CALL PUT1L

Output registers: CF=0 . . . . . Normal
CF=1 . . . . . Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

PUTBL (Put Block)

Function: Outputs the number of bytes of data indicated in the HL register to the file whose logical number is specified in the C register, starting at the address indicated in the DE register.

Input registers: The C register contains the logical number. The DE register contians the starting address of the data to be output. The HL register contains the number of bytes of data to be output.

Calling procedure: CALL PUTBL

Output registers: CF=0 . . . . . .Normal (DE ← the address following the end of the block output)
CF=1 . . . . . .Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC and IX. (Register HL is also preserved if CF=0)

PUTCR (Put Carriage Return)

Function: Outputs a carriage return to the file whose logical number is specified in the C register.

Input registers: The C register contains the logical number.

Calling procedure: CALL PUTCR

Output registers: CF=0 . . . . . Normal
CF=1 . . . . . Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

PUTM (Put Message)

PUTMX

Function: Outputs the line starting at the address indicated in register DE to the file whose logical number is specified in the C register. PUTM and PUTMX operate in the same manner except for their handling of $CRT and $LPT. Cursor control codes (, , etc.) are executed only when PUTM is used; when PUTMX is used, they are only display or printed as reverse characters. The end code (0DH) is not output.

Input registers: The C register contains the logical number. The DE register contains the starting address of the data to be output.

Calling procedure: CALL PUTM or CALL PUTMX

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

CLOSE (Close File)

KILL (Kill File)

Function: Closes or kills the file whose logical number is specified in the C register. If this subroutine is called when the C register contains 0, all currently opened files will be closed or killed. (This excludes files which were opened by FDOS itself.)

Input registers: The C register contains 0 or a logical number.

Calling procedure: CALL CLOSE or CALL KILL

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

LUCHK (LU Number Check)

Function: Checks whether a logical number (contained in the C register) has been defined.

Input registers: The C register contains the logical number.

Calling procedure: CALL LUCHK

Output registers: CF=1 ..... The logical number has not been defined.
CF=0 ..... The logical number has been defined.
L'←device number (see page 30 concerning device numbers)
IY←starting address of the device table. (see note 4 on page 13)

Registers preserved: All registers except AF, HL, IY, D'and L'.

Example:

```
LD    C,5            ;logical number
CALL  LUCHK
JP    C,NOTUSE
EXX
LD    A,L            ;device number
EXX
CP    4
JP    C,FD
:
```

DVINIT (Device Initialize)

Funciton: Initializes the write-opened device whose logical number is specified in the C register as follows.

Device	Operation
$CRT	line feed
$PTP	form feed
$LPT	form feed
$SOA, $SOB	normally output 0FH; this may be arbitrarily changed with the STATUS statement of FDOS.
$FDn, $CMT, $MEM	output a form feed code (0FH).

Output registers: The C register contains the logical number.

Calling procedure: CALL DVINIT

Output registers: CF=0 .....Normal
CF=1 .....Error (ACC←error code)
IY: Starting address of the device table (see note 4 on page 13)

Registers preserved: Only registers BC, DE, HL and IX.

MTOFF (Motor Off)

Function: Stops the motor of the floppy disk drive.
(The drive motor is activated automatically when necessary.)

Calling procedure: CALL MTOFF

Registers preserved: All registers except AF.

BREAK (Check Break Key)

Function: Checks whether [SHIFT] + [BREAK] have been pressed.

Input registers: None

Calling procedure: CALL BREAK

Output registers: CF=0 . . . . . Not pressed.
CF=1 . . . . . Pressed. (In this event, ACC ← 37 . 37 is the error code.)

Registers preserved: All registers except AF.

HALT (Halt Action with Break Action)

Function: Checks the keyboard and, if the space key is pressed, stops execution until the space key is pressed again. If [SHIFT] + [BREAK] are both pressed, ACC←37 and CF←1. (37 is the error code.)

Input registers: None

Call procedure: CALL HALT

Output registers: CF=0 . . . . . Normal
CF=1 . . . . . [SHIFT] + [BREAK] were pressed. (In this event, ACC←37.)

Registers preserved: All registers except AF.

SGETL (Screen Get Line)

Function: Inputs one line from the keyboard. The keyboard is equipped for auto repeat. The line which is actually input is the line in which the cursor is located when [CR] is pressed (the data may occupy two lines on the display screen). Afterwards, the mode is changed from the CAP/SML mode to the CAP mode.

Input registers: The DE register contains the starting address of the area (80 bytes required) in which the data is to be stored.

Calling procedure: CALL SGETL

Output registers: CF=0 ..... Normal
CF=1 ..... [SHIFT] + [BREAK] were pressed. ACC ←—0 (not 37)

Registers preserved: All registers except AF.

LTPNL (Let Printer New Line)

PMSGX (Printer Message X)

PMSG (Printer Message)

PPRNT (Printer Print)

PPAGE (Printer Page)

Function: These are printer control routines. These routines perform the same function for the printer as do the monitor subroutines shown below for the CRT.

	[Pinter]	[CRT]
1758	LTPNL (line feed)	LETNL
176C	PMSGX	MSGX
1771	PMSG	MSG
1791	PPRNT	PRNT
176F	PPAGE (form feed)	———

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ←— error code)

Registers preserved: All registers except AF and IY.

C& L1

&NL

&PRNT

&NMSG

&MSG

&1L

Function: Directs output to the printer or CRT depending on the presence or absence of the global switch ⌜ /P_⌟. &NL, &NMSG and &1L include the HALT function (see page 19 for the HALT function).

- C&L1 ..... Prepares either the printer or the CRT.
  This routine must be called before any other routines are used. Further, "?GSW" must be called before this routine is called.
- &NL ..... Performs the same function as LETNL.
- &PRNT ... Performs the same function as PRNT.
- &MSG .... Performs the same function as MSG.
- &NMSG ... Executes &NL, then executes &MSG.
- &1L ..... Executes &MSG, then executes &NL.

Output registers: CF=0 ..... Normal
CF=1 ..... Error (ACC ← error code)

Registers preserved: All registers except AF and IY.

See "LINKING ASSEMBLY PROGRAM WITH FDOS" in Appendix for an example of use.

PUSHR

PUSHR 2 (Push Register)

Function: This subroutine is used to PUSH registers IX, HL, DE and BC. (Only registers IX, HL and BC are pushed with PUSHR2.) The RET instruction at the end of this subroutine then automatically POPs these registers.

Calling procedure:

```
SUBR : CALL  PUSHR ; PUSH  REGISTERS
                .
                .
                .
       RET  Z          ; POP  and  RET  if  ZF = 1
                .
                .
       RET              ; POP  and  RET
```

Registers preserved: All registers except IX.

The program list shown below illustrates the function of this routine.

```
PUSHR  : ENT
         EX    (SP) , IX
         PUSH  HL
         PUSH  BC
         PUSH  DE
         PUSH  HL
         LD    HL ,POPR
         EX    (SP) , HL
         JP    (IX)
PUSHR2 : ENT
         EX    (SP) , IX
         PUSH  HL
         PUSH  BC
         PUSH  HL
         LD    HL ,POPR2
         EX    (SP) , HL
         JP    (IX)
POPR   : POP   DE
POPR2  : POP   BC
         POP   HL
         POP   IX
         RET
```

CHKACC (Check Acc)

Function: Checks whether the contents of ACC match any of several different given data items.

Input registers: ACC contains the data items to be checked.

Calling procedure:

```
CALL  CHKACC
DEFB  n          ; number of data items (1–255)
DEFB  data 1     }
DEFB  data 2     }
DEFB  data 3     }  n items of data to be compared
  :              }    DEFM '....'may be used with ASCII.
DEFB  data n     }
```

Output registers:
ZF=1 ..... One of the data items matches the contents of ACC.
ZF=0 ..... No match was found.

Registers preserved: All registers except the flags.

MULT (Multiply)

Function: Multiplies the contents of the DE register and the HL register (handling them as 16-bit unsigned integers) and places the result in the DE register.

Input registers: DE and HL

Calling procedure: CALL MULT

Output registers:
CF=1 ..... Overflow (result cannot be expressed in 16 bits)
CF=0 ..... Normal. The DE register contains the result of the calculation.

Registers preserved: All registers except AF, DE and HL.

SOUND (Warning Sound)

Function: Produces the sound "A0 □ ARA □ AR" to indicate that an error has occurred.

Calling procedure: CALL SOUND

Registers preserved: All registers.

BINARY (Convert ASCII to Binary)

Function: Converts an ASCII numeric string into a 16-bit unsigned integer.

Input registers: The HL register contains the starting address of the ASCII numeric string.

Calling procedure: CALL BINARY

Output registers: CF=1 ..... Overflow (cannot be expressed within 16 bits)
CF=0..... Normal. The DE register contains the converted data. The HL register contains the address following the end of the numeric string.
If the ASCII characters indicated by HL are not a numeric string, CF ←— 0 and DE ←— 0.

Registers preserved: All registers except AF, DE and HL.

Example:

```
        LD    HL,BUFFER
        CALL  BINARY
        JP    C,ERROR      ; if CF=0, DE becomes 400H.
          :                          HL points to 0DH.
BUFFER: DEFM  '1024'
        DEFB  0DH          ; must be an ASCII code for other than '0' ~ '9'
```

CASCII (Convert Binary to ASCII)

Function: Converts a 16-bit unsigned integer into an ASCII numeric string.

Input registers: The HL register contains the 16-bit unsigned integer. The DE register contians the address of the area in which the ASCII numeric string is to be stored.

Calling procedure: CALL CASCII

Output registers: The DE register contains the ending address of the ASCII numeric string obtained.

Registers preserved: All registers except AF and DE.

Example:

```
        LD    HL,1024
        LD    DE,BUFFER
        CALL  CASCII
          :
BUFFER: DEFS  10    ; after conversion the ASCII numeric string'1024' is stored.
```

CLEAR (Clear Area)

Function: Loads a continuous area in the memory with zeros. (The memory area must be 255 bytes or less.)

Input registers: None

Calling procedure:
```
CALL  CLEAR
DEFB  length      ; number of bytes to be cleared
DEFW  address     ; the memory is cleared starting at this address.
```

Output registers: None

Registers preserved: All registers.

CHLDE (Compare HL, DE)

Function: Compares the contents of the HL register with the contents of the DE register.

Input registers: HL and DE

Calling procedure: CALL CHLDE

Output registers: FLAG ← HL–DE; that is
CF = 0 , ZF = 0 ·····HL > DE
CF = 1 , ZF = 0 ·····HL < DE
CF = 0 , ZF = 1 ·····HL = DE

Registers preserved: All registers except AF.

LCHK (Limit Check)

Function: Compares the last usable memory area (the address indicated by the stack pointer minus 256) with the contents of the HL register.

Input registers: HL

Calling procedure: CALL LCHK

Output registers:
CF = 0 ·····HL < = SP – 256
CF = 1 ·····HL > SP – 256
At this time, ACC ←21. 21 is an error code. (memory protect error)

Registers preserved: All registers except AF.

ERR (Display Error Message)

Function: Displays an error message (see the System Error Messages in System Command for details). The contents of the C register and the IY register must be preserved from the time the error occurs until this routine is called. Further, the close or kill routine must not be called during that time (otherwise, the contents of the error message may be incorrect).

Input registers: The ACC register contains the error code (no error message is output if the error code is FFH).
The C register contains the logical number.
The IY register contains the starting address of the device table (see note 4 on page 13)
} These may not be necessary depending on the type of error. (see note 4 on page 13)

Calling procedure: CALL ERR

Output registers: ACC←FFH
CF←1

Registers preserved: All registers except AF.

Example:

```
CALL  SGETL     (page 20)
CALL  NC,&1L    (page 21)
JR    NC,-6
CALL  C,ERR
RET   C
```

ERRX

Function: This function displays a colon ":", followed by the contents of the area from the address following a specified 0DH until the next 0DH; the specified 0DH is the one which is the (ACC-1)th from the address indicated in the DE register.

Input registers: The DE register contains the starting address of the message block.
The ACC register contains a number (1-255).

Calling procedure: CALL ERRX

Output registers: ACC←FFH
CF←1

Registers preserved: All registers except AF.

Example:

```
ERMSG : DEFM  'SYNTAX'
        DEFB  0DH
        DEFM  'OVERFLOW'
        DEFB  0DH
        DEFM  'IL DATA'
        DEFB  0DH
        :
        LD    A,2
        LD    DE,ERMSG
        CALL  ERRX
```

This displays ':OVERFLOW'.

ERWAIT (Display Error Message and Wait Space Key)

Function:
1. Calls subroutine ERR if ACC≠0.
2. Displays the contents of the area starting with the address indicated in the DE register until 0DH.
3. Displays " , ⬇ SPACE KEY " if ACC=0.
4. Waits until [SPACE] or [SHIFT] + [BREAK] is pressed.

Input registers: ACC and DE

Calling procedure: CALL ERWAIT

Output registers: CF=0 ..... [SPACE] was pressed.
CF=1 ..... [SHIFT] + [BREAK] were pressed. (ACC ←37)

Registers preserved: All registers except AF.

—— FDOS Common Variables ——

LIMIT (Limit of Memory)

Number of bytes: 2

Meaning: Contains the last address plus one of RAM mounted.

ISTACK (Initial Stack Pointer)

Number of bytes: 2

Meaning: Contains the last address plus one of the memory area which is available to FDOS. This data is used by FDOS for initialization of the stack pointer. The contents of ISTACK may be changed by the FDOS LIMIT instruction. The contents of ISTACK must not be changed by any other means.

ZMAX

Number of bytes: 2

Meaning: Contains the last address of the area being used by FDOS (excluding the stack). The contents of ZMAX may be changed depending on the next subroutine called. (ROPEN, WOPEN, CLOSE, KILL, .CLI)

Caution: The area which may be used within the user program as free area is as follows.

1. [Lowest address] = [value contained in ZMAX when the user program was entered]
   + [number of files which are simultaneously opened (ROPEN or WOPEN)] x 350
   + [number of files which are simultaneously write-opened] x 72
   + [number of floppy disk units used] x 128

   [Maximum address] = [stack pointer (SP)] $-a$, a is approximately 256.
2. From ISTACK to LIMIT−1.
3. Area reserved by the DEFS statement within the assembly program.

.DNAME (Default File Name)

Number of bytes: 17

Meaning: The file name and succeeding 0DH contained in this area will be used as the default file name when the file name is omitted. For example, when this area contains "ABCD [CR] ", "$FD3" appearing on the command line will be interpreted as "$FD3 ; ABCD".

MDRIVE (Master Boot Drive)

Number of bytes: 1

Meaning: Contains the drive number minus one (0 ~ 3) of the drive containing the master diskette.

BDRIVE (Boot Drive)

Number of bytes: 1

Meaning: Contains the default drive number minus 1 (0 ~ 3). The default drive number is the number which appears to the left of the prompt " > " when FDOS is in the command wait state.

MAXDVR (Maximum Drive)

Number of bytes: 1

Meaning: Contains the number of floppy disk drives connected (1 ~ 4).

TODAY

Number of bytes: 7

Meaning: Contains the month, day and year followed by 0DH; each element of the date is indicated with a two-digit ASCII code.

RJOB (Running Job Pointer)

Number of bytes: 2

Meaning: Area which indicates how far command line interpretation has proceeded. When command lines are interpreted in a user program, the address following that of the last command line interpreted must be placed in RJOB.

—— CLI Intermediate Code Table ——

Switch

ASCII	Intermediate code
	80 H
/D	81 H
/C	82 H
/E	83 H
/G	84 H
/L	85 H
/N	86 H
/S	87 H
/P	88 H
/O	89 H
/T	8AH
	8BH
/LF	8CH
/PN	8DH
/PO	8EH
/PE	8FH

Device name

ASCII	Device number	Intermediate code
$ FD1	0	90 H
$ FD2	1	91 H
$ FD3	2	92 H
$ FD4	3	93 H
$ CMT	4	94 H
$ MEM	5	95 H
	6	96 H
	7	97 H
	8	98 H
	9	99 H
$ PTR	10	9AH
	11	9BH
$ KB	12	9CH
$ SIA	13	9DH
$ SIB	14	9EH
	15	9FH

ASCII	Device number	Intermediate code
$ LPT	16	A0 H
$ PTP	17	A1 H
$ CRT	18	A2 H
	19	A3 H
$ SOA	20	A4 H
$ SOB	21	A5 H
	22	A6 H
	23	A7 H
$ USR1	24	A8 H
$ USR2	25	A9 H
$ USR3	26	AAH
$ USR4	27	ABH
	28	ACH
	29	ADH
	30	AEH
	31	AFH

Built-in commands

ASCII	Intermediate code
RUN	B0H
DATE	B1H
XFER	B2H
DIR	B3H
PAGE	B4H
RENAME	B5H
DELETE	B6H
TYPE	B7H
CHATR	B8H
FREE	B9H
BYE	BAH
TIME	BBH
EXEC	BCH
HCOPY	BDH
	BEH

File mode

ASCII	File mode number	Intermediate code
.*	255	F0H
		F1H
.OBJ	1	F2H
.BTX	2	F3H
	3	F4H
.ASC	4	F5H
.RB	5	F6H
	6	F7H
.LIB	7	F8H
	8	F9H
	9	FAH
.SYS	10	FBH
	11	FCH
	12	FDH
	13	FEH
	14	FFH

Other Codes other than those shown in this table are expressed as is in ASCII code. However, this applies only to 01H-7FH. The codes for some small characters and graphic characters are the same as CLI intermediate codes; therefore, they cannot be used.

BASIC RELOCATABLE LIBRARY (RELO. LIB)

The BASIC relocatable library contains a collection of subroutines which are required by programs created using the BASIC compiler. These routines are useful when BASIC program subroutines (external functions, external commands, etc.) are created using the assembler.

Routines contained in RELO. LIB can only be used as BASIC subroutines; they cannot be executed as independent assembly programs.

. . INT 0

. . INT 1

. . INT 2 (Convert Floating to Fixed)

Function: Converts a real number expressed in 5 bytes into a 16-bit integer. The absolute value of any decimal fraction is discarded.
(Examples: 1.5 ⟶ 1 −2.7 ⟶ −2)
. .INT 0 . . . . . The input range is from −32768 ~ 32767
. .INT 1 . . . . . The input reange is from 0 ~ 255
. .INT 2 . . . . . The input range is from −32768 ~ 65535

Input registers: The HL register contains the starting address of the 5 byte real number.

Calling procedure: CALL . . INTO CALL . . INT1 CALL . . INT2

Output registers: HL ⟵ integer

Error processing: . . INT 0 . . . . . Upon overflow, CF ⟵ 1.
. . INT 1 . . . . . Upon overflow, JP ⟵ ER3.
. . INT 2 . . . . . Upon overflow, CF ⟵ 1.

Registers preserved: All registers except AF and HL.

Note: The . . FLT0 and CONST subroutines (described below) are used to create the 5-byte real number.

. . FLT 0 (Convert Fixed to Floating)

Function: Converts a 16-bit signed integer into a 5-byte real number.

Input registers: The HL register contains the 16-bit signed integer. The DE register contains the starting address of the area in which the real number is stored.

Calling procedure: CALL .. FLT0

Registers preserved: All registers except AF, DE and HL.

CASC'**(Change ASCII)**

Function: Converts a 16-bit unsigned integer into an ASCII character string and appends 0DH to the end of it.

Input registers: The HL register contains the 16-bit unsigned integer. The DE register contains the starting address of the area in which the ASCII character string is stored.

Calling procedure: CALL CASC'

Registers preserved: All registers except AF.

MSGS (High Speed Message)

Function: Performs the same function as the monitor subroutine MSG, but at high speed.

Registers preserved: All registers except AF.

.MOVE'**(Move String)**

Function: Converts a character string from type 1 to type 2. The converted character string is stored in an area called .WORD. (The type 1 and type 2 character string formats are explained on page 35.)

Input registers: The HL register contains the starting address of the character string (in type 1.)

Calling procedure: CALL .MOVE'

Output registers: The DE register contains the starting address of the converted character string. (The address of .WORD)

Registers preserved: All registers except AF, BC, DE and HL.

FASCX (Convert Floating to ASCII)

Function: Converts a 5-byte real number into an ASCII character string and appends 0DH to the end of it.

Input registers: The HL register contains the starting address of the real number. The DE register contains the starting address of the area in which the ASCII character string is stored.

Calling procedure: CALL FASCX

Registers preserved: None

CONST (Convert ASCII to Const)

Function: Converts a constant expressed in ASCII code into a 5-byte real number.

Input registers: The HL register contains the starting address of the constant expressed in ASCII code. The DE register contains the starting address of the area in which the result is stored.

Calling procedure: CALL CONST

Output registers: The HL register contains the first address following the constant converted.

Registers preserved: None

Error processing: JP ER3

CHCOND (Character Condition)

Function: Compares the two character strings (in type 1).

Input registers: The HL and DE registers contain the starting addresses of each of the two character strings being compared.

Calling procedure: CALL CHCOND

Output registers: FLAG ← (DE) − (HL)
that is,
CF=0, ZF=0 ····· (DE) > (HL)
CF=1, ZF=0 ····· (DE) < (HL)
CF=0, ZF=1 ····· (DE) = (HL)

Registers preserved. All registers except AF, BC, DE and HL.

ER1 ER13

ER2 ER14

ER3 ER21

ER4 ER24

ER5 ER37

ER6 ER64

Function: Error message display routine used during BASIC program execution. See the Error Message table in the BASIC compiler instruction manual (available separately).

Calling procedure: JP ER1 (SYNTAX ERROR), etc.

BEERR (Basic Executing Error)

Function: Error message dispaly routine used during BASIC program execution.

Calling procedure:
```
CALL  BEERR
DEFB    error code (error number in BASIC)
DEFM    'ERROR MESSAGE'
DEFB    0DH
```
→ No return made.

BABORT (Basic Abort)

Function: When a system error occurs during BASIC program execution, this routine displays the applicable error message and interrupts execution.

Input registers: The ACC register contains the error code (system error number).
The C register is the logical number.
The IY register contains the starting address of the device table (see note 4 on page 13).
} May not be required depending upon the type of error.

Calling procedure:
```
JP    BABORT
```

Example:
```
LD    C,2
CALL  GETlC
JP    C,BABORT
```

Caution: BEERR is a routine which displays ∗ER nn: message in linenumber (where nn is the error number in BASIC) when an error occurs in BASIC program; BABORT is a routine which displays −ERR message in linenumber when an error occurs at the FDOS level. ON ERROR processing will be performed in both cases, if specified.

. .STOP

Function: Stops BASIC program execution. (Corresponds to the STOP instruction of the BASIC compiler.)

Calling procedure: JP . .STOP

. . . END

Function: Terminates BASIC program execution. (This corresponds to the END instruction of the BASIC compiler.)

Calling procedure: JP . . .END

.WORD

Function: 257-byte general purpose area.

——Type 1 and Type 2 Character String Formats——

There are two types of character strings which are handled by BASIC; these should be used as appropriate.

Type 1

```
DEFB length    (character string length: 0 ~ 255)
DEFM  '......'
```

Type 2

```
DEFM  '......'
DEFB 0DH
```

INDEX OF LIBRARY NAMES

Type:
- MON ..... Monitor subroutine
- CLI ..... Related to CLI
- IOCS ..... Related to IOCS
- UTYL ..... Utility
- VAR ..... FDOS common variable
- RELO ..... BASIC relocatable library

} FDOS subroutines

Appendix

SHARP

NOTICE

The MZ-80 series of sophisticated personal computers is manufactured by the SHARP CORPORATION. Hardware and software specifications are subject to change without prior notice; therefore, you are requested to pay special attention to version numbers of the monitor (supplied in the form of ROM) and the system software (supplied in the form of cassette tape or mini-floppy disk files).

This manual is for reference only and the SHARP CORPORATION will not be responsible for difficulties arising out of inconsistencies caused by version changes, typographical errors or omissions in the descriptions.

This manual is based on the monitor SP-1002 and the SP-7000 series FDOS.

—— CONTENTS ——

LINKING ASSEMBLY PROGRAM WITH FDOS

An object program generated with the FDOS editor, assembler and relocating loader can be executed with the RUN command.

Example 1 1 > RUN GALAXY CR

This command loads GALAXY.OBJ into memory from the floppy disk and executes it. Execution of a RET statement in the object program returns control to FDOS. The contents of the stack pointer must be restored to the value contained when the object program was called before the RET statement is executed. CF must be reset before control is returned because an error message will be output on the assumption that the ACC contains a system error code if CF is set.

Global switches and/or arguments can be assigned after the file name in the RUN command as shown below.

Example 2 2 > RUN ASMZ8/P CONTROL-A CR

Global switch

Argument

In this case, FDOS converts the entire command line into intermediate codes (refer to the Library/Package Manual), and loads ASMZ8.OBJ into memory from the floppy disk, then executes it. At this time, the HL register points to the intermediate code corresponding to /P (88H). The RJOB area in FDOS has the same value as the HL register.

Switches and arguments following the file name (ASMZ8) must be decoded by user object program. They can be decoded using FDOS subroutines. When the last character (":" or 0DH) is decoded, the HL register contents must be stored in RJOB. To return control to FDOS, execute a RET statement in the object program.

The sample program listed on the following pages illustrates command line decoding. It outputs an ASCII file to the CRT display or printer. This program operates in a manner similar to the FDOS TYPE command. The file name of this program is TYPE'. Thus, executing

Example 3 2 > RUN TYPE' ABC CR

Outputs file ABC.ASC to the CRT display and executing

Example 4 2 > RUN TYPE'/P ABC CR

Outputs ABC.ASC to the printer.

All external labels (indicated by the E message) in this program list are defined in FDOS.LIB.

——SAMPLE PROGRAM (COMMAND)——

```
   **  Z80 ASSEMBLER SP-7101  PAGE 01  **

.
01 0000                    ; TYPE COMMAND
02 0000                    ;
03 0000                    .TYPE:  ENT
04 0000 116200                     LD    DE,SWTBL          ; DE:=SWITCH TABLE
05 0003 CD0000      E              CALL  ?GSW              ; CHECK GLOBAL SWITCH
06 0006 D8                         RET   C
07 0007 CD0000      E              CALL  C&L1              ; SELECT CRT OR LPT
08 000A CD0000      E              CALL  ?SEP              ; CHECK SEPARATER
09 000D D8                         RET   C
10 000E FE2C                       CP    2CH               ; SEPARATER="," ?
11 0010 3E03                       LD    A,3               ;   3 IS ERR CODE
12 0012 37                         SCF
13 0013 C0                         RET   NZ                ; NO, ERR RETURN
14 0014 CD0000      E     TYPE0:   CALL  ?LSW              ; CHECK LOCAL SWITCH
15 0017 D8                         RET   C
16 0018 3E08                       LD    A,8               ;   8 IS ERR CODE
17 001A 37                         SCF
18 001B C0                         RET   NZ                ; ERROR, LSW EXIST
19 001C 0E80                       LD    C,128             ; LU#:=128
20 001E D9                         EXX
21 001F 0604                       LD    B,4               ; DEFAULT MODE=ASC
22 0021 D9                         EXX
23 0022 CD0000      E              CALL  ROPEN             ; READ-OPEN
24 0025 D8                         RET   C
25 0026 CD0000      E              CALL  &NL
26 0029 382D                       JR    C,TYPEER
27 002B CD0000      E              CALL  TESW              ; TEST GLOBAL SWITCH
28 002E 88                         DEFB  88H               ; /P
29 002F 3F                         CCF
30 0030 DC0000      E              CALL  C,PPAGE           ; LPT PAGING
31 0033 3823                       JR    C,TYPEER
32 0035 CD0000      E              CALL  MODECK            ; FILEMODE CHECK
33 0038 04                         DEFB  4                 ;  .ASC ?
34 0039 116400                     LD    DE,BUFFER
35 003C D40000      E     TYPE10:  CALL  NC,GET1L          ; GET 1 LINE
36 003F D40000      E              CALL  NC,&1L            ; DISP OR PRINT 1 LINE
37 0042 30F8                       JR    NC,TYPE10         ; NO ERROR
38 0044 A7                         AND   A
39 0045 C25800                     JP    NZ,TYPEER         ; ERROR
40 0048                    ;                              ; END-OF-FILE
41 0048 CD0000      E     TYPE20:  CALL  CLOSE            ; CLOSE FILE
42 004B CD0000      E              CALL  ?SEP              ; CHECK SEPARATER
43 004E D8                         RET   C
44 004F FE2C                       CP    2CH               ; SEPARATER="," ?
45 0051 28C1                       JR    Z,TYPE0           ; YES, TYPE NEXT FILE
46 0053 220000      E              LD    (RJOB),HL         ; SAVE CLI POINTER
47 0056 AF                         XOR   A
48 0057 C9                         RET
49 0058 CD0000      E     TYPEER:  CALL  ERR               ; ERROR OCCUR
50 005B CD0000      E              CALL  KILL              ; KILL FILE (C=128)
```

```
01 005E 37                          SCF                     ; SET ERROR FLAG
02 005F 3EFF                        LD      A,FFH           ; ALREADY DISP ERR MSG
03 0061 C9                          RET
04 0062 88                  SWTBL:  DEFB    88H             ; /P
05 0063 FF                          DEFB    FFH             ; END OF SWTBL
06 0064                     BUFFER: DEFS    128             ; 128 BYTE BUFFER
07 00E4                             END
```



```
.TYPE  0000  BUFFER 0064  SWTBL  0062  TYPE0  0014  TYPE10 003C
TYPE20 0048  TYPEER 0058
```

USER CODED I/O ROUTINES

FDOS supports control programs not only for the floppy disk drive but for the printer ($LPT) and the paper tape reader ($PTR), etc. Other I/O devices can be operated under the control of FDOS by means of user coded control programs.

—USER I/O ROUTINE—

A user I/O routine consists of the following sections.

A. Device table (57 bytes)
B. ROPEN or WOPEN procedure
C. Data transfer program
D. CLOSE and KILL procedures

These sections are explained below using the FDOS paper tape reader control program ($PTR) as an example.

A. Device Table (lines 2 through 20, bytes 0 through 56)

*FDOS uses bytes 0 through 2 (FLAG0 – FLAG2).
This area must be written exactly as it is shown.

*Byte 3 (flag 3) represents the attribute of the I/O device.

Bit 7 : 0

Bit 6 : 1 indicates that tabulation is possible. This bit is set to 1 for the printer. See note 1 on page 7.)

Bit 5 : 1 indicates that parity specification ($PTR/PE, etc.) can be made.

Bit 4: 1 indicates that only .ASC-mode files can be transferred.

Bit 3: 0

Bit 2: 0

Bit 1: 1 indicates that WOPEN is possible. (See note 2 on page 7).

Bit 0 : 1 indicates that ROPEN is possible. (See note 2 on page 7).

*Byte 4 indicates the data transfer format. (described later)

*Bytes 5 and 6 are the starting address of the subroutine to be called during ROPEN execution.

*Bytes 7 and 8 are the starting address of the subroutine to be called during WOPEN execution.
(WOPEN is not executed for $PTR so DEFW 0 is specified in this program.)

*Bytes 9 and 10 are data by the FDOS STATUS command. (Not used for $PTR).

*Bytes 11 through 14 are starting addresses of the subroutines for CLOSE and KILL processing.

*Bytes 15 through 22 (Procedures 1 through 4) are data transfer routine addresses. The data transfer procedure differs according to the transfer format.

ROPEN

Transfer format	1	4
Procedure 1	Input 1 character (ACC ← data)	Input 1 line (From the address indicated by DE to a CR code.)
Procedures 2 ~ 4	Unused	Unused

WOPEN

Transfer format	1	5
Procedure 1	Unused	Carriage return
Procedure 2	Output 1 character (ACC : data) On.ASC mode, 0DH means carriage return and 0CH means form feed.	Output 1 character (ACC : data) On . ASC mode, 0DH means carriage return and 0CH means form feed.
Procedure 3	Unused	Output 1 line (Corresponds to monitor subroutine MSG)
Procedure 4	Unused	Output 1 line (Corresponds to monitor subroutine MSGX)

* Bytes 23 and 24 are used by FDOS.
* Byte 25 is used only when bit 6 of FLAG 3 is 1, in which case it must be loaded with the number of characters of the line which have been output by I/O routine.
* Byte 26 is loaded with the file mode by FDOS.
* Bytes 27 through 56 are the device name (up to 16 characters); the rest area must be reserved with DEFS.
* When the transfer format is 4, a buffer area for 1 line is reserved after the byte 56 with DEFS.

B. ROPEN or WOPEN procedure (lines 22 through 30)

Only ROPEN is needed for the paper tape reader ($PTR). The tape feeder is skipped by this procedure. WOPEN is also used to start a new page during output of an assembly listing.

C. Data transfer program (lines 32 through 63)

Program which performs actual transfer of data.

D. CLOSE and KILL procedures (lines 65 through 66)

No function with $PTR.

To return control to FDOS from the ROPEN, WOPEN, Procedure 1–4, CLOSE or KILL routines, set registers as follows before executing the RET statement.

Normal : CF ← 0

Error : CF ← 1, ACC ← error code (refer to the System Error Messages in the System Command Manual.)

File end : CF ← 1, ACC ← 0

The contents of the IY, BC′, DE′ and HL′ registers must be saved in any case.

—— RELOCATING USER I/O ROUTINES ——

First, assemble the program coded (the program name DVM is used below).

Example 1: 2 > ASM DVM, $LPT/L [CR]

Next, relocate the file to generate the object program. A higher loading address must be specified at this time because of factors related to the LIMIT command described later. Take care to ensure that addresses do not over lap when two or more user I/O programs are used. If necessary, link MON.LIB or FDOS.LIB with the user I/O programs.

Example 2 : 2 > LINK $C000, DVM [CR]
Example 2′ : 2 LINK $C400, CDISP, $FD1; FDOS.LIB [CR]

—— LINKING USER I/O ROUTINES WITH FDOS ——

User I/O routines must be linked with FDOS I/O controller every time FDOS is activated.

First, use the LIMIT command to reserve an area in memory for loading the object program (DVM. OBJ).

Example 3 : 2 > LIMIT $C000 [CR]

Next, load the object program.

Example 4 : 2 > LOAD DVM [CR]

Finally, link the routine to the FDOS I/O controller. $USR1 through 4 are provided in FDOS as device names for user I/O routines; assign the user I/O control routine to one of these device names.

Example 5 : 2 > ASSIGN $USR1, $C000 [CR]

Now the user program is linked with FDOS and can be called by specifying $USR1 (–4). It is convenient to prepare EXEC files which include LIMIT, LOAD and ASSIGN commands such as those shown above. (Refer to the System Command Manual).

User I/O programs are called as shown below.

Example of use by FDOS commands

2 > TYPE $USR1 [CR]
2 > XFER DATA4, $USR2 [CR]

Example of use by BASIC compiler

```
 10  ROPEN #2, "$USR1"
 20  INPUT #2, A$
 30  IF EOF (#2) THEN 100
 40  PRINT A$
      :
      :
999  CLOSE #2
```

Note 1: Bit 6 determines the functions of BASIC statements PRINT # and INPUT #.
When bit 6 = 1, data is treated in the same manner as with PRINT and INPUT statements.
When bit 6 = 0, separators (",", and ";") in the PRINT # statement are replaced with [CR] and commas included in the input character string for the INPUT # statement are treated as data; only [CR] is regarded as a data separator. (This is the same as with the PRINT # statement supported by SP-6015 and the PRINT/T statement supported by SP-5025.)

Note 2: Both ROPEN and WOPEN are possible, when both bit 1 and bit 0 are set, but they cannot be executed simultaneously.

——SAMPLE PROGRAM (I/O DRIVER)——

```
   **  Z80 ASSEMBLER SP-7101  PAGE 01  **


01 0000                    $PTR:    ENT
02 0000 0000                        DEFW  0                 ; FLAG 0,1
03 0002 00                          DEFB  0                 ; FLAG 2
04 0003 21                          DEFB  21H               ; FLAG 3
05 0004 01                          DEFB  1                 ; TRANSFER FORMAT
06 0005 3900                        DEFW  ROPEN             ; ROPEN
07 0007 0000                        DEFW  0                 ; WOPEN
08 0009 0000                        DEFW  0                 ; STATUS
09 000B 7D00                        DEFW  CLC               ; CLOSE
10 000D 7D00                        DEFW  CLC               ; KILL
11 000F 4700                        DEFW  PTR1C             ; PROCEDURE 1
12 0011 0000                        DEFW  0                 ; PROCEDURE 2
13 0013 0000                        DEFW  0                 ; PROCEDURE 3
14 0015 0000                        DEFW  0                 ; PROCEDURE 4
15 0017                             DEFS  2
16 0019 00                          DEFB  0                 ; TAB
17 001A 00                          DEFB  0                 ; FILEMODE
18 001B 24505452                    DEFM  '$PTR'            ; FILENAME
19 001F 0D                          DEFB  0DH
20 0020                             DEFS  25
21 0039                    ;
22 0039 CD5400             ROPEN:   CALL  PTRIN             ;ROPEN
23 003C D8                          RET   C
24 003D 78                          LD    A,B
25 003E A7                          AND   A
26 003F 28F8                        JR    Z,ROPEN
27 0041 78                          LD    A,B
28 0042 327F00                      LD    (DATA),A
29 0045 AF                          XOR   A
30 0046 C9                          RET
31 0047                    ;
32 0047 CD5400             PTR1C:   CALL  PTRIN
33 004A 3E00                        LD    A,0
34 004C D8                          RET   C
35 004D 217F00                      LD    HL,DATA
36 0050 7E                          LD    A,M
37 0051 70                          LD    M,B
38 0052 A7                          AND   A
39 0053 C9                          RET
40 0054                    ;
41 0054 3EEF               PTRIN:   LD    A,EFH
42 0056 D3E1                        OUT   (E1H),A
43 0058 DBE1               PTR2:    IN    A,(E1H)
44 005A CB6F                        BIT   5,A
45 005C 2017                        JR    NZ,PTR8
46 005E CB67                        BIT   4,A
47 0060 28F6                        JR    Z,PTR2
48 0062 DBE1               PTR3:    IN    A,(E1H)
49 0064 CB6F                        BIT   5,A
50 0066 200D                        JR    NZ,PTR8
```

```
   **  Z80 ASSEMBLER SP-7101  PAGE 02  **


01 0068 CB67                        BIT   4,A
02 006A 20F6                        JR    NZ,PTR3
03 006C DBE0                        IN    A,(E0H)
04 006E 2F                          CPL
05 006F 47                          LD    B,A
06 0070 3EFF                        LD    A,FFH
07 0072 D3E1                        OUT   (E1H),A
08 0074 C9                          RET
09 0075 3EFF              PTR8:     LD    A,FFH
10 0077 D3E1                        OUT   (E1H),A
11 0079 37                          SCF
12 007A 3E3C                        LD    A,60              ;NOT READY
13 007C C9                          RET
14 007D                   ;
15 007D AF                CLC:      XOR   A
16 007E C9                          RET
17 007F                   ;
18 007F                   DATA:     DEFS  1
19 0080                             END
```

```
   **  Z80 ASSEMBLER SP-7101  PAGE 03  **


$PTR   0000  CLC    007D  DATA   007F  PTR1C  0047  PTR2   0058
PTR3   0062  PTR8   0075  PTRIN  0054  ROPEN  0039
```

CONVERSION OF CASSETTE BASED SYSTEM PROGRAMS

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.

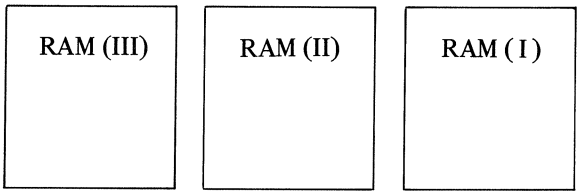
DESTROY FDOS?

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.

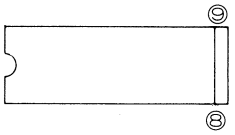
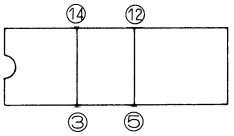
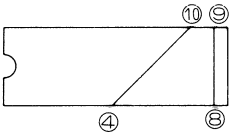
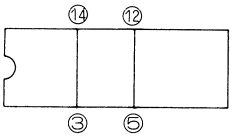
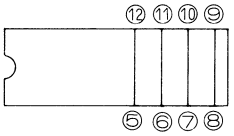
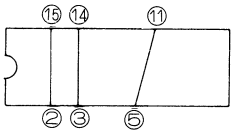
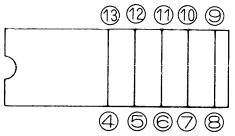
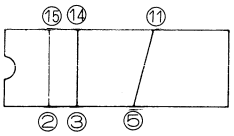
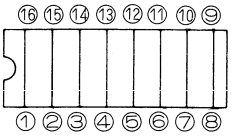
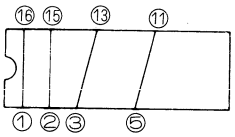
MEMORY EXPANSION

FDOS requires 36K bytes of RAM

The MZ-80K has three RAM blocks and RAM chips of up to 16K bytes can be mounted on each block.



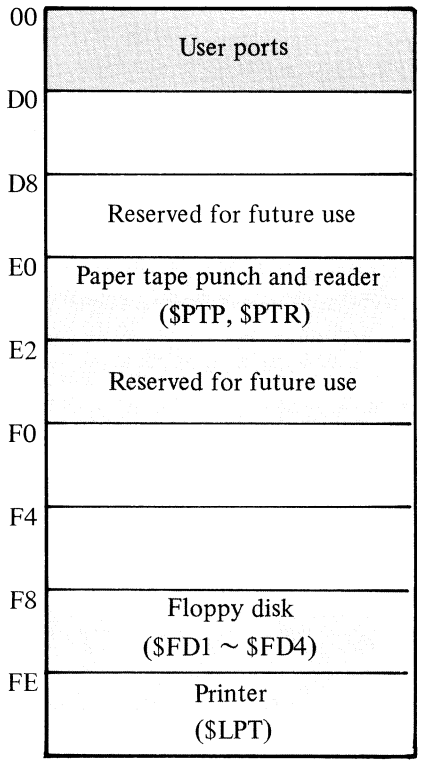
The following table shows the procedure for connecting address sectors in the address range for the RAM area.

RAM chips mounted	RAM area address range	Address sector connections	
		CS1	CS2
Eight 4K dynamic RAM chips on the RAM (II) block (MZ-80K standard)	1000 ∫ 5FFF		
Eight 4K dynamic RAM chips on both RAM (II) and RAM (III) blocks	1000 ∫ 6FFF		
Eight 16K dynamic RAM chips on the RAM (II) block	1000 ∫ 8FFF		
Eight 16K dynamic RAM chips on the RAM (II) block and eight 4K dynamic RAM chips on the RAM (III) block	1000 ∫ 9FFF		
Eight 16K dynamic RAM chips on both RAM (II) and RAM (III) blocks	1000 ∫ CFFF		

In all cases, eight 16K dynamic RAM chips are assumed to be mounted on the RAM (I) block.

I/O MAP

I/O ports with addresses equal to or higher than DO_H are reserved by the manufacturer for control of external devices; those used by FDOS are assigned device names such as $LPT.



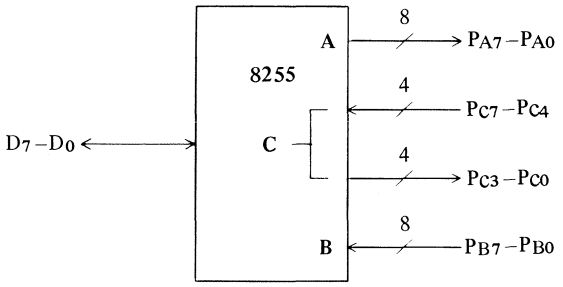
MEMORY MAPPED I/O

Memory addresses E000H through E003H are assigned to the programmable peripheral interface (8255) and E004H through E008H are assigned to the programmable interval timer (8253).

The monitor program sets the mode for 8255 and 8253 as follows.

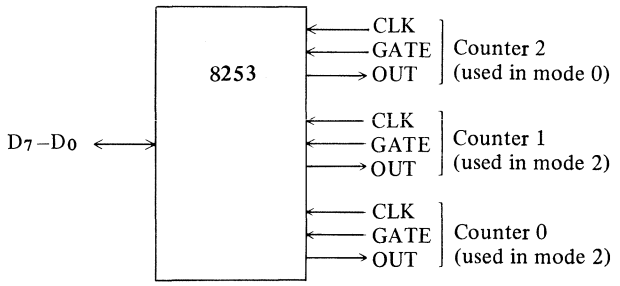
Control word

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	1	0



Control word

D7	D6	D5	D4	D3	D2	D1	D0	
0	1	1	1	0	1	0	0	Sets counter 2 to one second (constant value).
1	0	0	0	0	0	0	0	Reads counter 2.
0	0	1	1	0	1	0	0	Sets the frequency division ratio.



8255 port definition (mode 0)

A		B		Group A			Group B	
D4	D3	D2	D1	Port A	Port C (upper 4 bits)	#	Port B	Port C (lower 4 bits)
0	0	0	0	Output	Output	0	Output	Output
0	0	0	1	Output	Output	1	Output	Input
0	0	1	0	Output	Output	2	Input	Output
0	0	1	1	Output	Output	3	Input	Input
0	1	0	0	Output	Input	4	Output	Output
0	1	0	1	Output	Input	5	Output	Input
0	1	1	0	Output	Input	6	Input	Output
0	1	1	1	Output	Input	7	Input	Input
1	0	0	0	Input	Output	8	Output	Output
1	0	0	1	Input	Output	9	Output	Input
1	0	1	0	Input	Output	10	Input	Output
1	0	1	1	Input	Output	11	Input	Input
1	1	0	0	Input	Input	12	Output	Output
1	1	0	1	Input	Input	13	Output	Input
1	1	1	0	Input	Input	14	Input	Output
1	1	1	1	Input	Input	15	Input	Input

8253 control word format

D7	D6	D5	D4	D3	D2	D1	D0
SCI	SC0	RLI	RL0	M2	M1	M0	BCD

SCI	SC0	Counter selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Unused

RLI	RL0	Read/load selection
0	0	Latch the counter contents
0	1	Read/load MSB
1	0	Read/load LSB
1	1	Read/load LSB, then MSB

M2	M1	M0	Mode selection
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

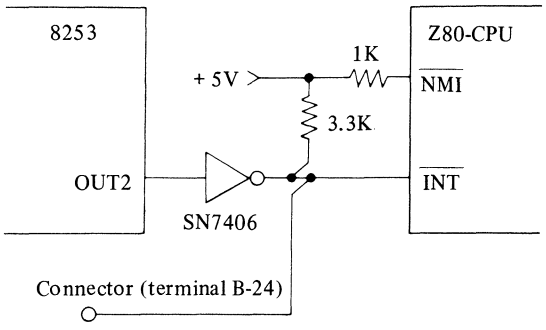
BCD	Binary/BCD selection
0	16 bit binary counter
1	4 digit BCD counter

MEMORY MAPPED I/O CHART

Address (hex.)	Memory read	Memory write
E000		D_7 — Cursor blinking timer reset signal D_3 to D_0 — Keyboard scan row output signals
E001	D_7 to D_0 — Keyboard scan column input signals	
E002	D_7 — $\overline{\text{V-BLANK}}$ D_6 — Cursor blinking timer status bit D_5 — Read data from the cassette tape deck D_4 — Record/play key signal from the cassette tape deck	D_3 — Cassette deck motor ON/OFF signal D_2 — CAP/SML lamp selection signal D_1 — Write data to the cassette deck D_0 — $\overline{\text{V-GATE}}$
E003		8255 C port single bit set/reset CAP (green) lamp—LD A, 05H LD (E003H), A SML (red) lamp – LD A, 04H LD (E003H), A
E004		8253 counter 0 setting data The music signal is generated by dividing the 2MHz clock signal with counter 0 in the MZ-80K. Monitor subroutine CALL MSTA sets the division ratio and controls the gates.
E005	8253 counter 1 read data	8253 counter 1 preset data
E006	8253 counter 2 read data	8253 counter 2 preset data
E007		8253 mode setting data
E008	D_0 – Tempo timer status bit	D_0 – Music Start (1), Stop (0)

NOTES FOR USE OF INTERRUPT

The Z80-CPU $\overline{INT}$ terminal is connected to the B-24 terminal of the connector provided on the rear panel as shown at right. It is recommended that an open collector driver be used to feed the $\overline{INT}$ signal externally.



The non-maskable interrupt terminal (NMI) is internally connected to the +5 V line through a 1 kΩ resistance.

The MZ-80K monitor program sets the CPU to interrupt response mode 1 and disables the interrupt.

—— EXECUTION OF TI$ IN BASIC AND TIME IN FDOS ——

The BASIC interpreter developed for the MZ-80K supports TI$ and the FDOS supports TIME to read the built-in timer. The built-in timer uses counters 1 and 2 of 8253. It is set to 00:00:00 when the BASIC interpreter or FDOS is started. Interrupts are enabled at the same time as the timer is set. The level at terminal OUT2 of 8253 becomes high every 12 hours to cause an interrupt and make a 24 hour clock on the program.

When the CPU accepts an interrupt, it automatically executes a restart to location 0038H. Instruction JP 1038H is stored in 0038H and an instruction causing a jump to the clock updating program is stored in 1038H. Therefore, an externally applied $\overline{INT}$ signal cannot be identified and it is also treated as an interrupt for the built-in clock. Thus, the $\overline{INT}$ signal cannot be used externally when the BASIC interpreter or the FDOS is operating unless the clock function is disabled.

—— INTERRUPT BY USER PROGRAM ——

In an FDOS user program, use of the $\overline{INT}$ signal is optional. The EI instruction must be executed properly since the monitor program disables the interrupt.

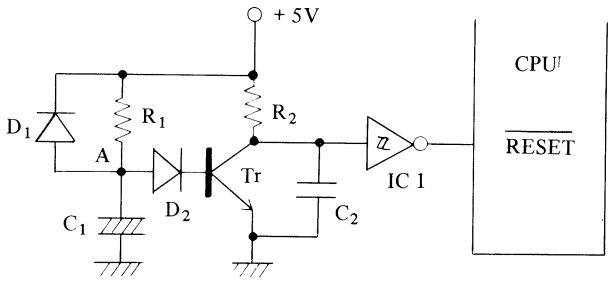
8253 has no reset function, but the equivalent of a reset is performed when a control word is set without presetting the counter to mask the interrupt from OUT2.

HARDWARE RESET

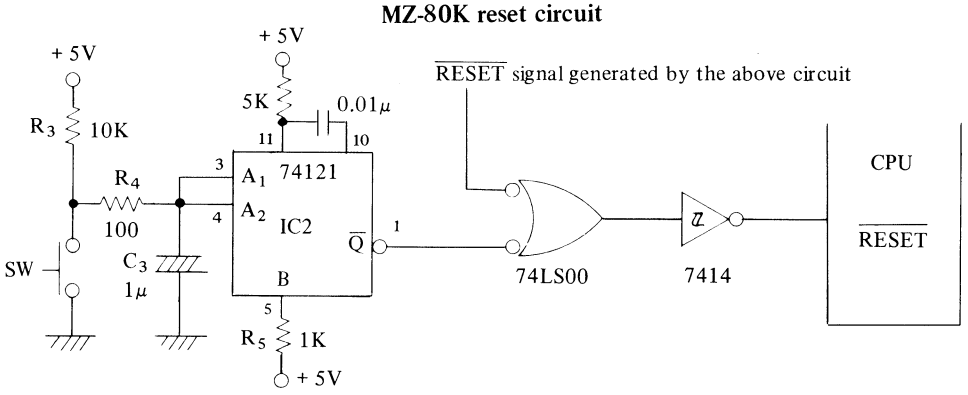
Microcomputer operating conditions are generally indeterminate when the power is turned on. The RESET signal is used to set the microcomputer to a certain initial operating state.

When the RESET signal is applied, the program counter is set to 0 and the monitor program stored in locations 0000H and on is activated to start operation of the MZ-80K.

In the MZ-80K, the reset signal is automatically applied by the circuit shown below when the power is turned on.



MZ-80K reset circuit



Reference circuit for manual reset

The base voltage of Tr is low when the power is turned on and it is kept low until C1 is charged. During this interval, Tr is off and its collector level is high. This high level signal is inverted by IC1 to apply the $\overline{\text{RESET}}$ signal to the CPU.

Tr is turned on when the voltage at point A reaches V_{BE} of Tr plus foward voltage V_D of D2 so that the $\overline{\text{RESET}}$ signal is turned off. When the power is turned off, C1 is discharged through D1. Therefore, the reset circuit operates normally for a few seconds after the power is turned off.

It is often necessary to reset the CPU without turning off the power. For example, the system runs out of control when the POKE statement is erroneously used in a BASIC program and the BASIC interpreter is destroyed. The power must be turned off to restart the system; this takes a rather long time because the data and program must be loaded again.

The above problem can be solved by adding a manual reset circuit; this circuit is shown abvoe. IC2 is an one-shot multivibrator. When SW is turned on, the voltage at A1 and A2 drops to a low level and a negative pulse is generated at $\overline{Q}$. This negative pulse is used for resetting the CPU.

Note that the program error is not corrected by resetting the CPU and so the program may run without control again. This manual reset circuit is provided for reference only.

PAPER TAPE PUNCH AND READER INTERFACE

FDOS has built-in paper tape punch and reader control programs. These are assigned the device names $PTP and $PTR, respectively. In actuality, however an interface circuit must be established with a universal interface I/O card to connect the paper tape puch and reader with the MZ-80 series microcomputer. The circuit diagram is shown on page 37.

The method for controlling the paper tape punch and reader is not standardized. A paper tape punch and reader which can be controlled by FDOS must have the following signal timing system. The signal names and timing charts shown below are based on the RP-600 paper tape punch and reader manufactured by Nada Electronics Laboratory. (For details, refer to the manual included with the paper tape punch and reader.)

—— SIGNAL NAME ——

Puncher

$\overline{DT_1} \sim \overline{DT_8}$: Data (PTP ← CPU)

$\overline{MI}$※ : Motor ON/OFF control signal (PTP ← CPU)

$\overline{ST}$: START/STOP control signal (PTP ← CPU)

$\overline{TO}$: Timing signal (PTP → CPU)

($\overline{RDY}$)※※ : Ready state signal (PTP → CPU)
(This signal is not output from the RP-600 since it can be used in remote operation. Ground it when the RP-600 is used.)

Reader

$\overline{RD_1} \sim \overline{RD_8}$: Data (PTR → CPU)

$\overline{STA}$: START/STOP control signal (PTR ← CPU)

$\overline{SPR}$: Sprocket signal (PTR → CPU)

$\overline{RB}$: Tape end signal (abnormal stop signal) (PTR → CPU)

※ Do not connect when the motor is not remotely controlled.

※※ The DPT26A manufactured by the Anritsu Electric Co. outputs this signal, but the RP-600 does not.

—— I/O PORTS ——

Port $E0_H$ is used for data by both the punch and the reader. Port $E1_H$ is used for control signals. See Table 1.

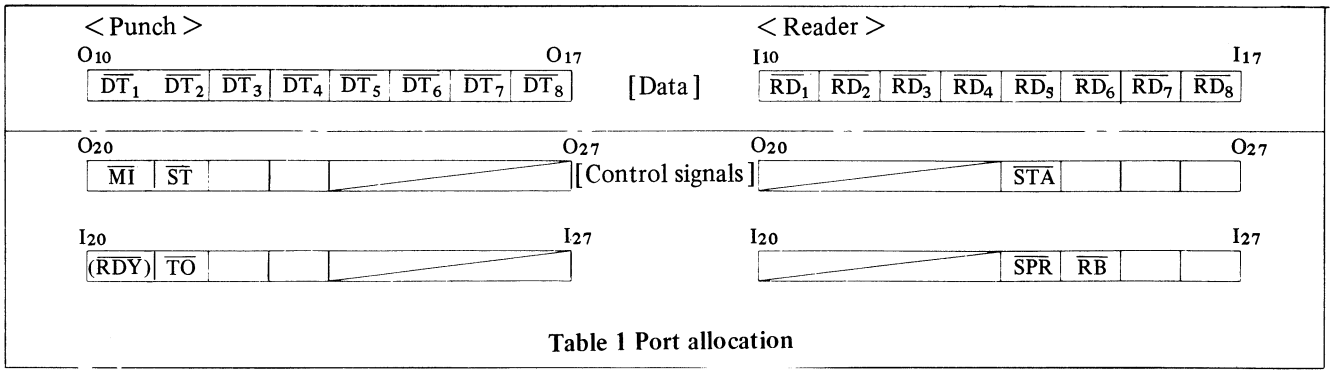


Table 1 Port allocation

——TIMING CHART——

Punch

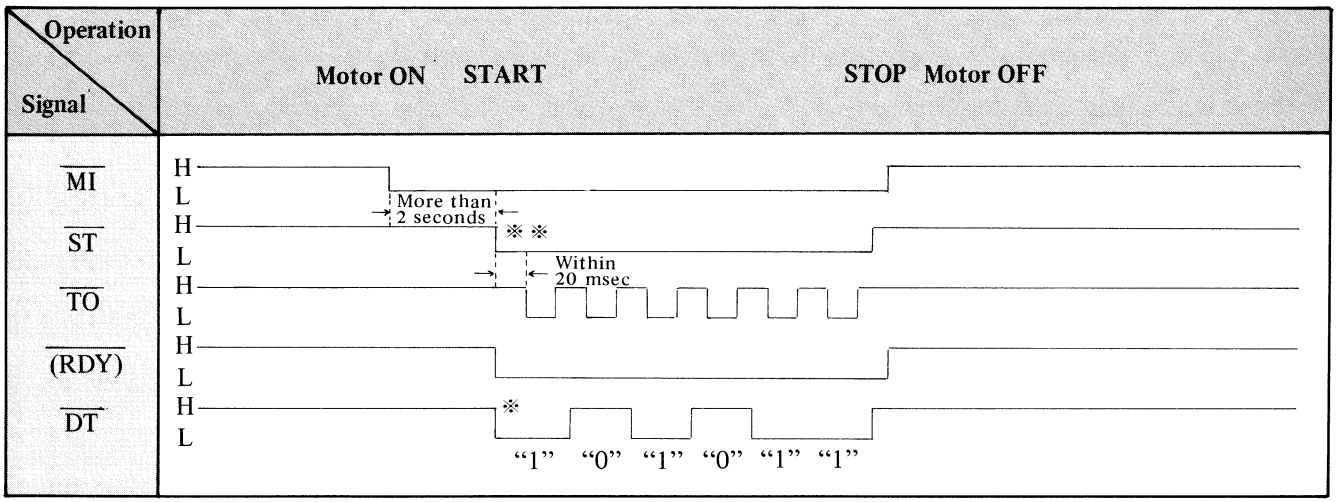


Figure 1 Punch timing chart

* The next data to be punched is readied while $\overline{TO}$ is H and maintained while $\overline{TO}$ is L.

** $\overline{ST}$ is set to L 2 or more seconds after the motor has been started, and is set to H after $\overline{TO}$ has risen from L to H for the last data.

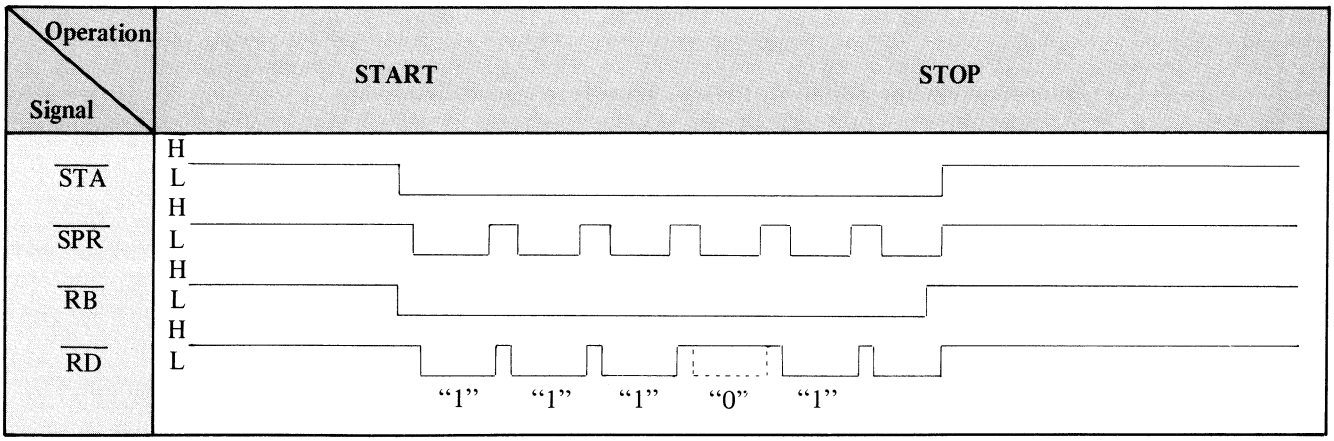


Figure 2 Reader timing chart

—— PREPARING A PAPER TAPE PUNCH/READER I/O CARD ——

It is convenient to use a universal I/O card (MZ-80 I/O-1) for preparing a paper tape punch and reader I/O interface circuit. Markings such as 0_{10} or 0_{17} in the port allocation table on page 18 match those on the universal I/O card.

See page 20 for setting the universal I/O card switches to select port addresses E0 and E1.

The RP-600 internal interface circuit and input and output pin connections are shown below for reference. (For details, refer to the manual included with the RP-600).

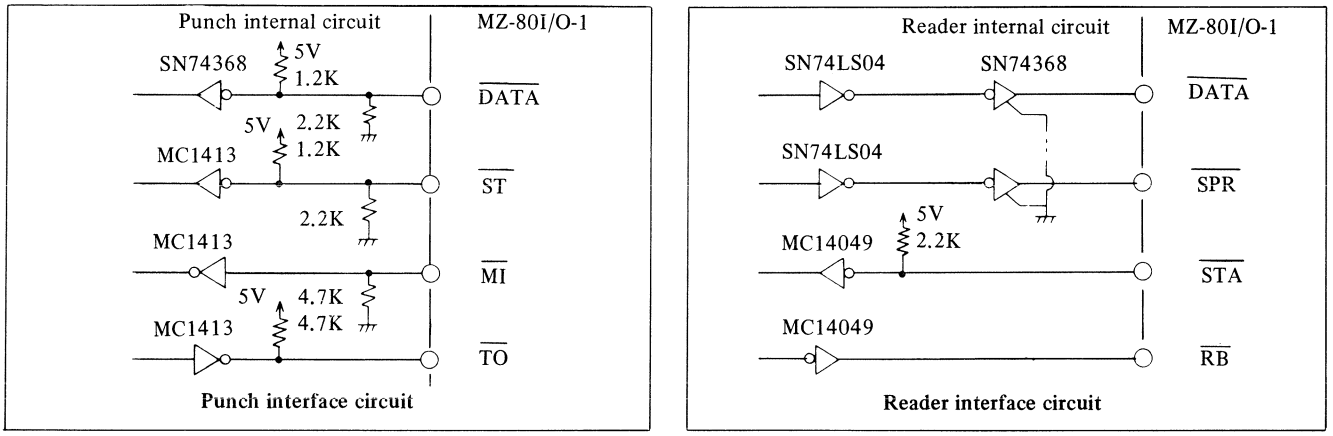


Figure 3 Interface circuit (RP-600)

Pin	Signal		Pin	Signal	
1	$\overline{DT_1}$	Data	13		
2	$\overline{DT_2}$		14		
3	$\overline{DT_3}$		15		
4	$\overline{DT_4}$		16		
5	$\overline{DT_5}$		17		
6	$\overline{DT_6}$		18		
7	$\overline{DT_7}$		19		
8			20		
9	$\overline{DT_8}$		21		
10			22	$\overline{MI}$	Motor ON/OFF signal
11	$\overline{TO}$	Timing signal	23	$\overline{ST}$	START/STOP signal
12	GND		24	FG	Frame ground

Pin	Signal		Pin	Signal	
1	$\overline{RD_1}$	Data	19		
2	$\overline{RD_2}$		20		
3	$\overline{RD_3}$		21		
4	$\overline{RD_4}$		22		
5	$\overline{RD_5}$		23		
6	$\overline{RD_6}$		24		
7	$\overline{RD_7}$		25		
8	$\overline{SPR}$	Sprocket signal	26		
9	$\overline{RD_8}$	Data	27		
10			28	$\overline{STA}$	START/STOP signal
11			29	$\overline{RB}$	Operating state signal
12	GND		30	FG	Frame ground
13			31		
14			32		
15			33		
16			34		
17			35		
18			36		

Table 2 Connector pin connections

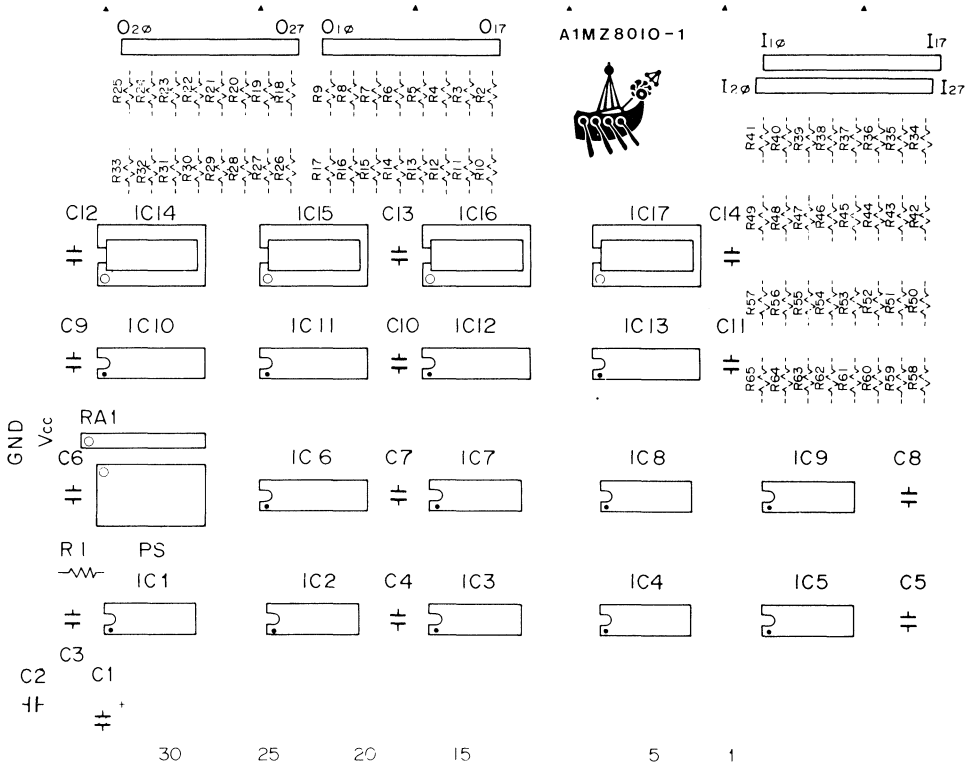


Figure 4 Universal I/O card component location (parts)

(1) Number of ports

Input : 2 ports Output : 2 ports

(2) Port address

All port addresses can be set. (However, FDOS uses DO_H and higher locations.)

The input port for $I_{10} \sim I_{17}$ is set to an even address.

The input port for $I_{20} \sim I_{27}$ is set to an odd address.

The output port for $O_{10} \sim O_{17}$ is set to an even address.

The output port for $O_{20} \sim O_{27}$ is set to an odd address.

(3) Port address setting switches (PS)

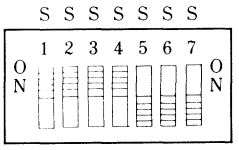
Numbers marked on the PS switches correspond to the address bus lines shown below. Turning a PS switch OFF sets the corresponding address bit to logical "1" and turning it ON to logical "0".

Switch No.	7	6	5	4	3	2	1
Address bit	AB_7	AB_6	AB_5	AB_4	AB_3	AB_2	AB_1

Example) Setting the PS switches as shown below sets the port address to EO_H.

1	1	1	0	0	0	0	0
↑	↑	↑	↑	↑	↑	↑	
S_7	S_6	S_5	S_4	S_3	S_2	S_1	

S_7 ..... OFF
S_6 ..... OFF
S_5 ..... OFF
S_4 ..... ON
S_3 ..... ON
S_2 ..... ON
S_1 ..... ON



When the PS switches are set as shown above, ports EO_H and $E1_H$ are used for this card.

CAUTION: Installing two or more interface cards which have the same port address settings will result in destruction of ICs.

Universal I/O card port address setting

BUILDING MONITOR PROGRAMS AND CHARACTER GENERATORS

A monitor program and character generator which meet the requirements of a particular user system can be built by the user. However, an expert knowledge of software and hardware is required to build these because software resources (the BASIC interpreter, system program, application program, etc.) and hardware resources (printer, disk device, etc.) supplied by the Sharp Corporation may not be used with a carelessly generated monitor program.

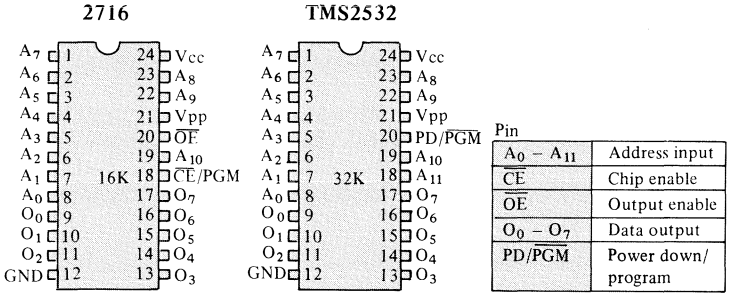
The MZ-80K uses the following LSIs for the monitor and character generator.

Mask ROM type	
Monitor	: X0171PA (equivalent to μPD2332) x 1
C.G	: X0172PA (equivalent to μPD2316E) x 1

Mask ROM LSIs μPD2332 and μPD2316E are pin compatible with PROM LSIs TMS2532 and 2716, respectively. Pin connections are shown below. Be careful not to insert an LSI chip in a wrong socket since all LSI chips are of the 24-pin DIP type. Place LSI chips in a conductive rubber mat when storing them. (LSI chips destroyed by inproper handling are not guaranteed.)

—— PRECAUTIONS FOR HANDLING LSI CHIPS ——

- Ground the human body when handling LSI and IC chips. Avoid wearing clothing made of chemical fiber.
- Hold LSI or IC chips by the plastic resin housing and avoid directly touching the pins.
- Pay attention to the direction in which LSI and IC chips are inserted into their sockets.
- When storing an LSI or IC chip, wrap it in aluminum foil or place it in a conductive rubber holder to keep all pins at the same potential.
- Do not store LSI and IC chips in extremely wet or dry places. The storage temperature is −20°C ~ +70°C for LSIs and −40°C ~ +125°C for ICs, but store them at a normal temperature if possible.
- Do not subject LSI and IC chips to shock and do not apply excessive pressure to the pins.
- Be sure to turn the power off before removing LSI and IC chips from thier sockets.



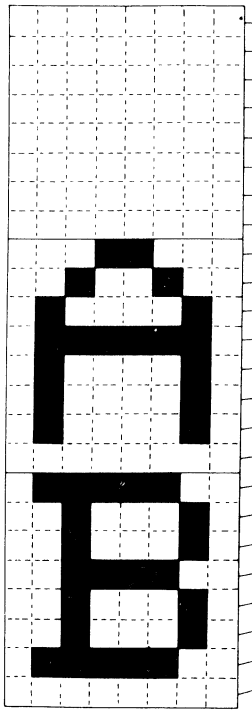
Pin	
$A_0 - A_{11}$	Address input
$\overline{CE}$	Chip enable
$\overline{OE}$	Output enable
$O_0 - O_7$	Data output
PD/$\overline{PGM}$	Power down/ program

—— MONITOR ——

FDOS can be used to develop user monitor programs. Refer to published information (various magazines, guide books, manuals, etc.) for the structure of ordinary monitor programs. The circuit diagrams included will be useful in the development of user monitor programs.

—— CHARACTER GENERATOR ——

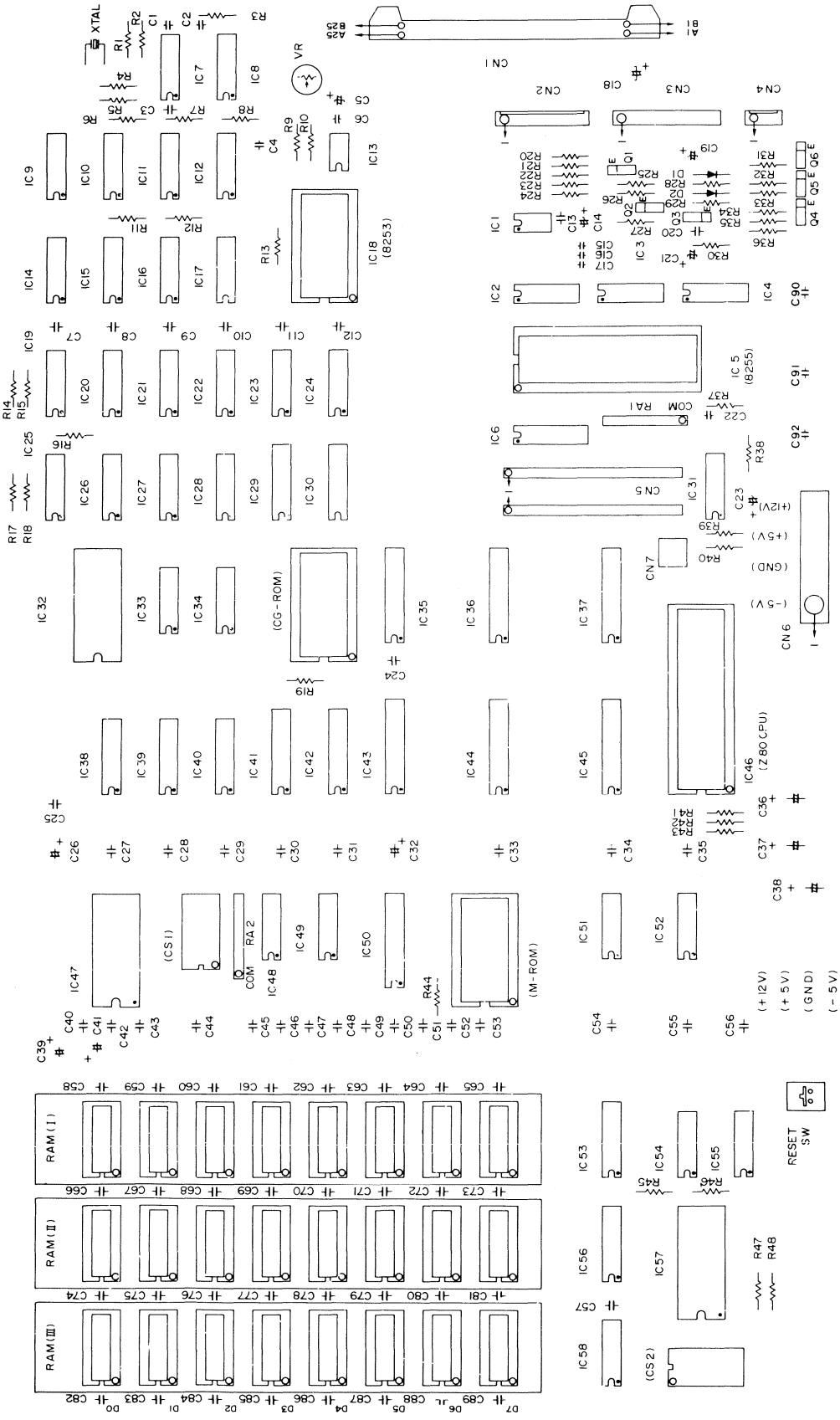
A character generator must store 8 x 8 character patterns corresponding to display codes 00H through FFH. The following sample shows the relationship between the storage format of a space, A and B and the assembly listing. Assemble the texts for all desired characters and operate the PROM formatter to make a character generator.



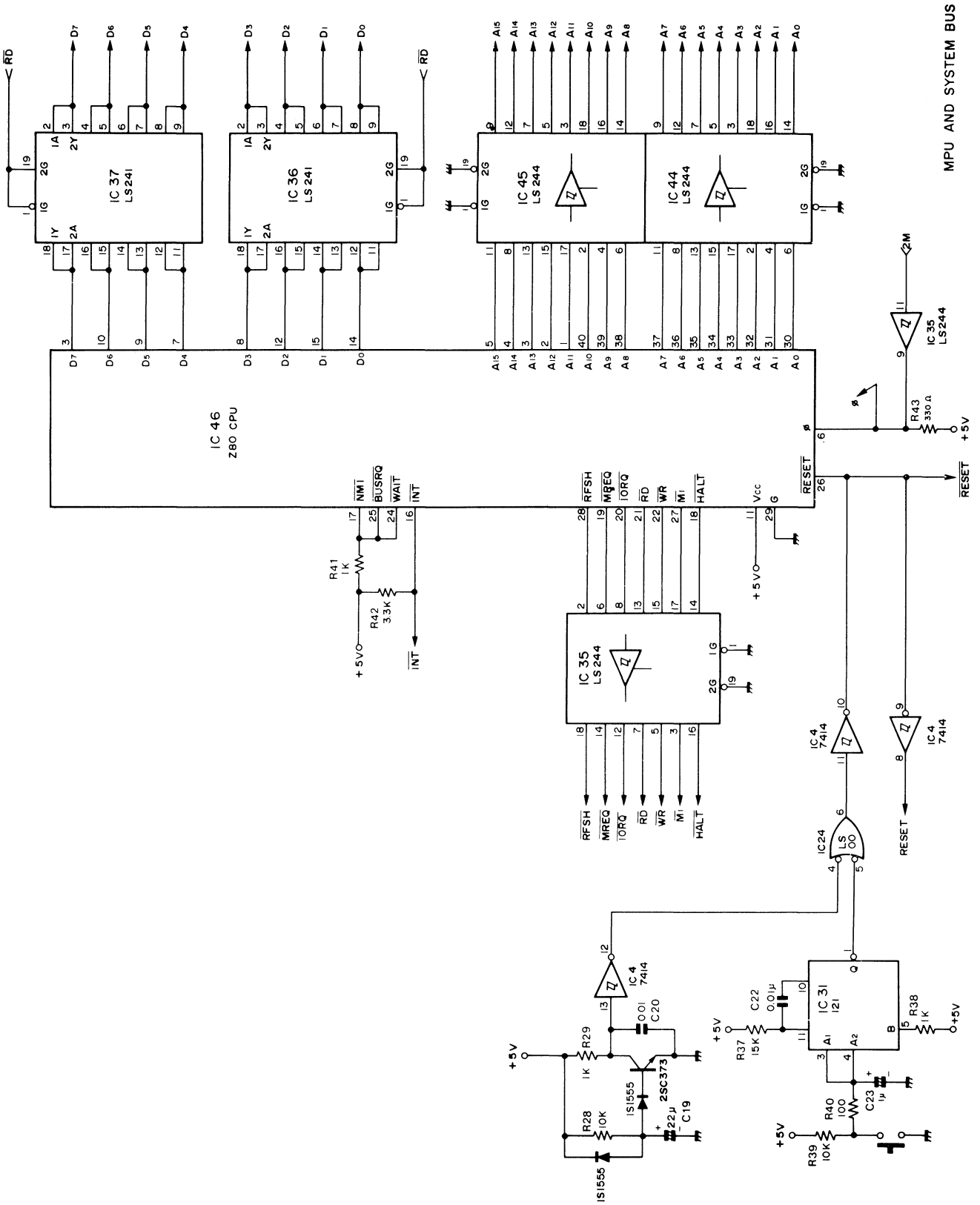
```
**  Z80 ASSEMBLER SP-7101 PAGE 01  **
01 0000 00      DEFB  0         ;SPACE
02 0001 00      DEFB  0
03 0002 00      DEFB  0
04 0003 00      DEFB  0
05 0004 00      DEFB  0
06 0005 00      DEFB  0
07 0006 00      DEFB  0
08 0007 00      DEFB  0
09 0008 18      DEFB  18H       ;A-PATTERN
10 0009 24      DEFB  24H
11 000A 42      DEFB  42H
12 000B 7E      DEFB  7EH
13 000C 42      DEFB  42H
14 000D 42      DEFB  42H
15 000E 42      DEFB  42H
16 000F 00      DEFB  0
17 0010 7C      DEFB  7CH       ;B-PATTERN
18 0011 22      DEFB  22H
19 0012 22      DEFB  22H
20 0013 3C      DEFB  3CH
21 0014 22      DEFB  22H
22 0015 22      DEFB  22H
23 0016 7C      DEFB  7CH
24 0017 00      DEFB  0
25 0018         END
```

MZ-80K CIRCUIT DIAGRAMS

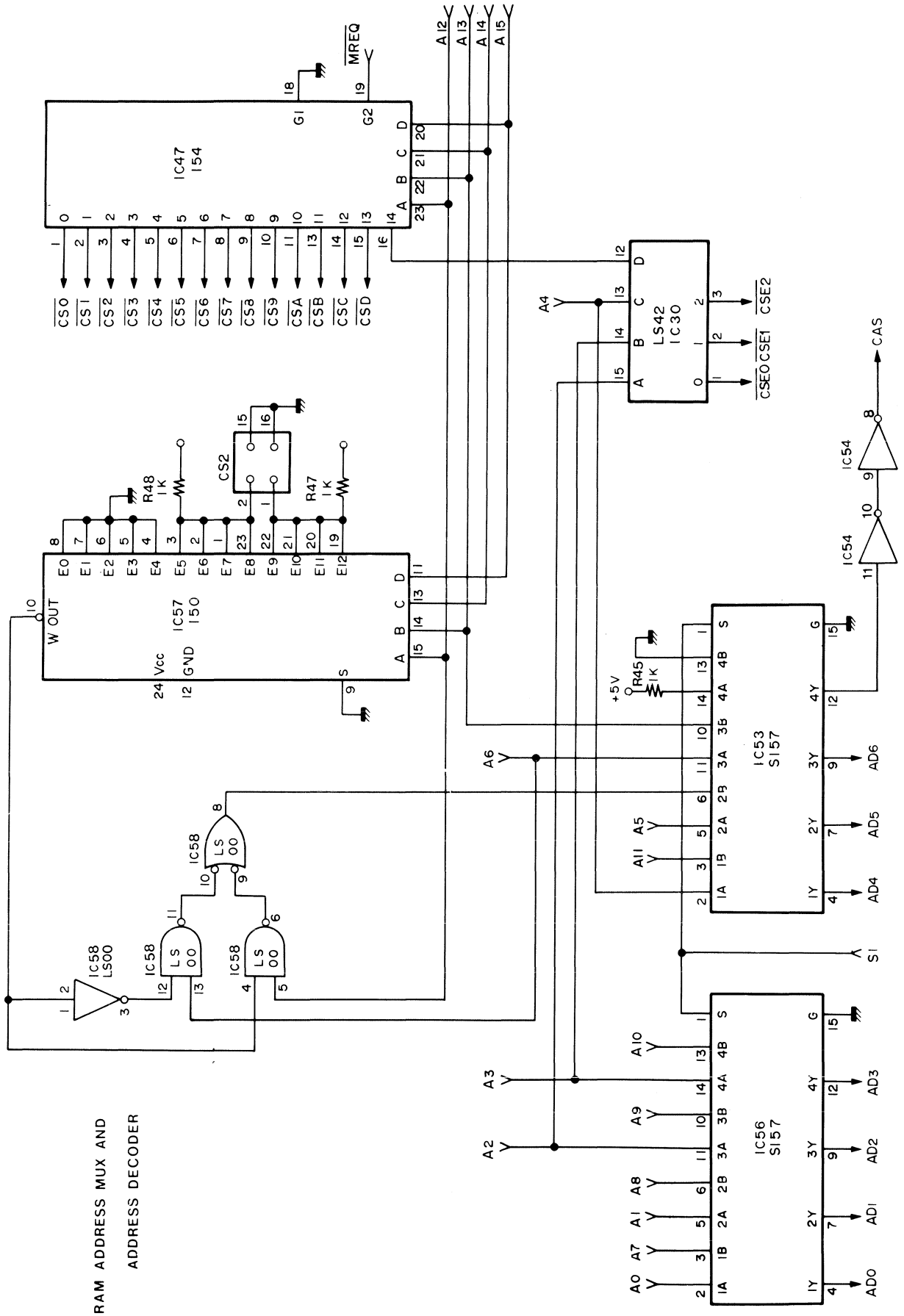
■ CPU Card Component Locations



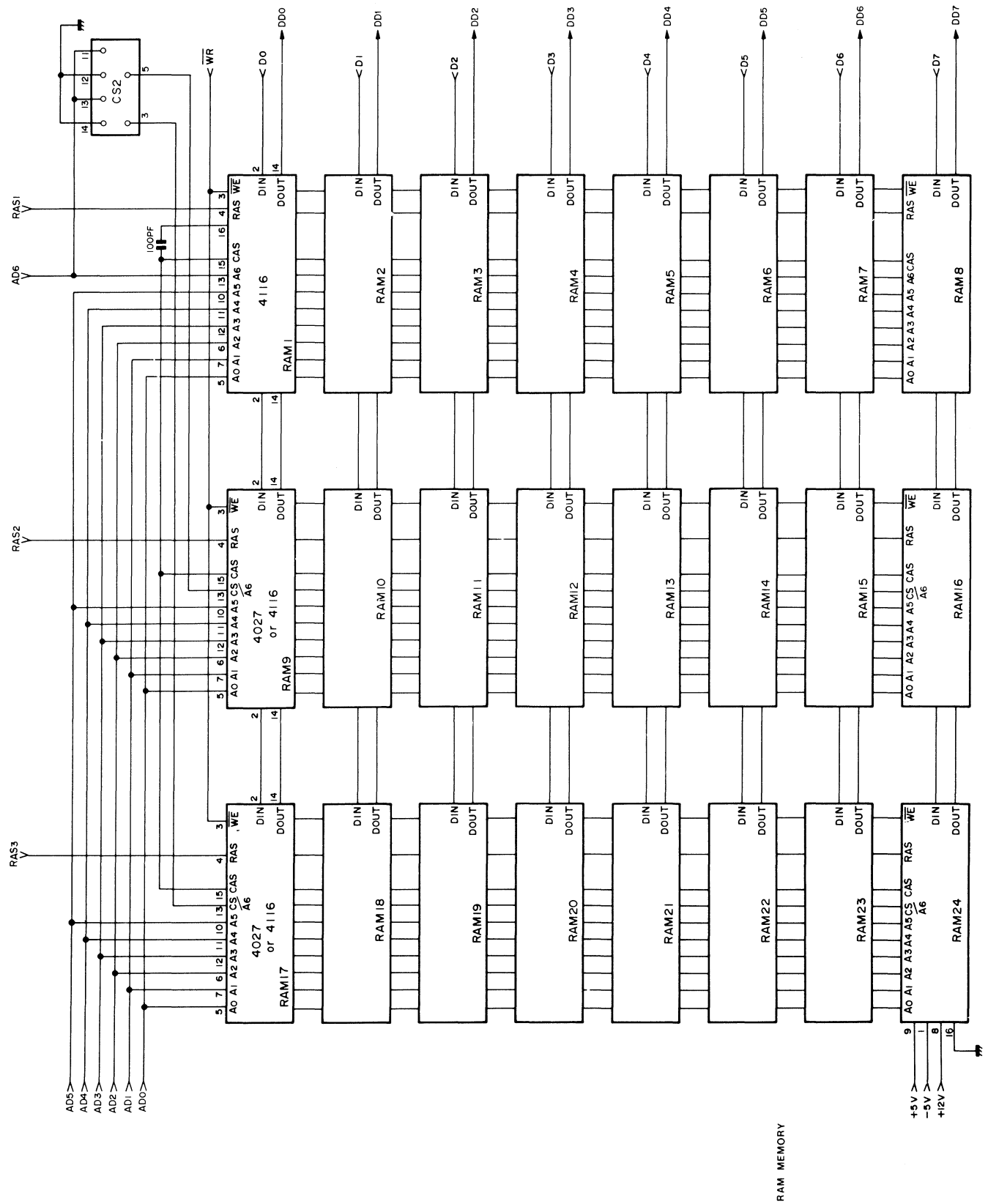
■ CPU Circuit Diagram (1)



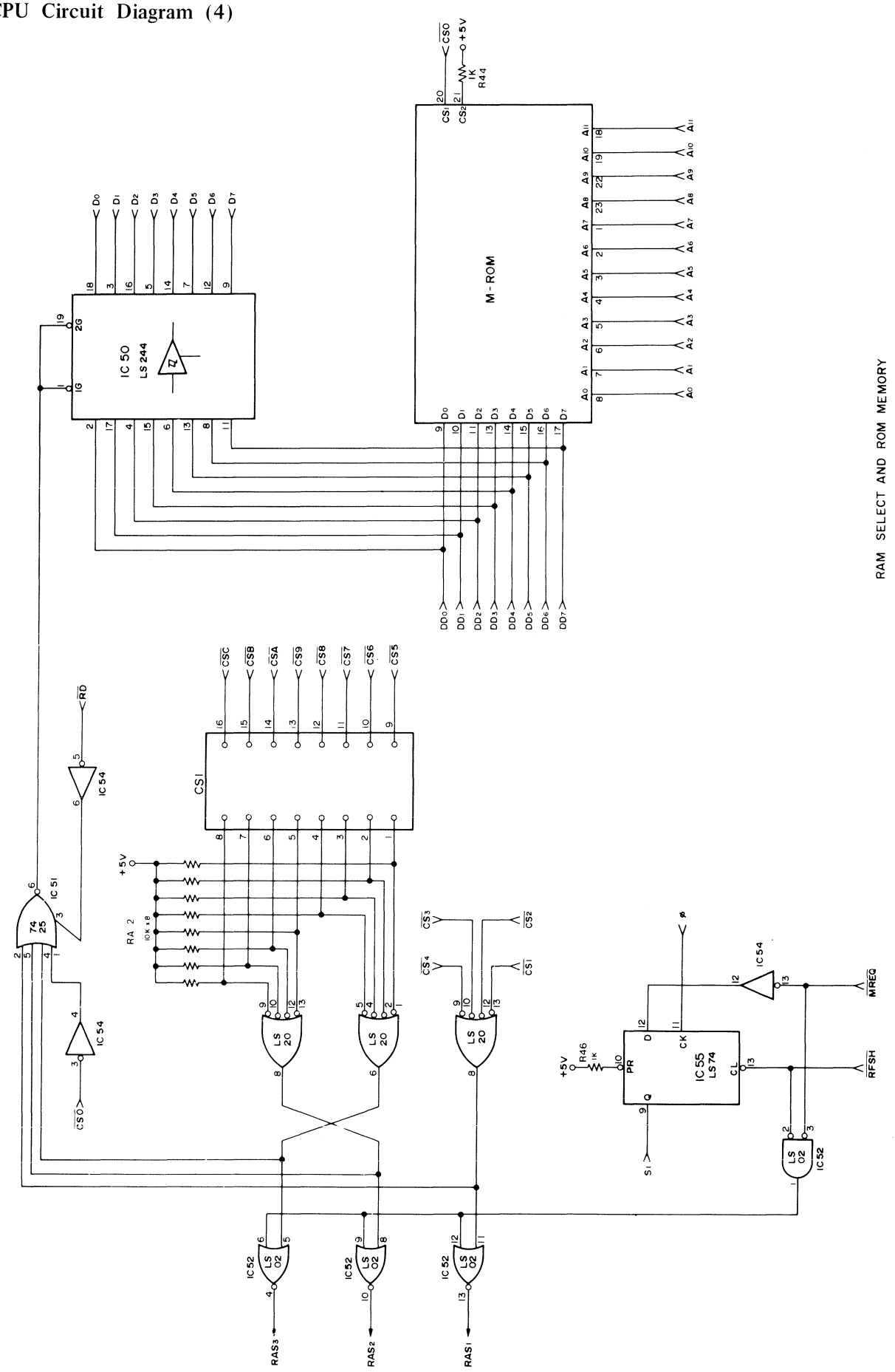
■ CPU Circuit Diagram (2)



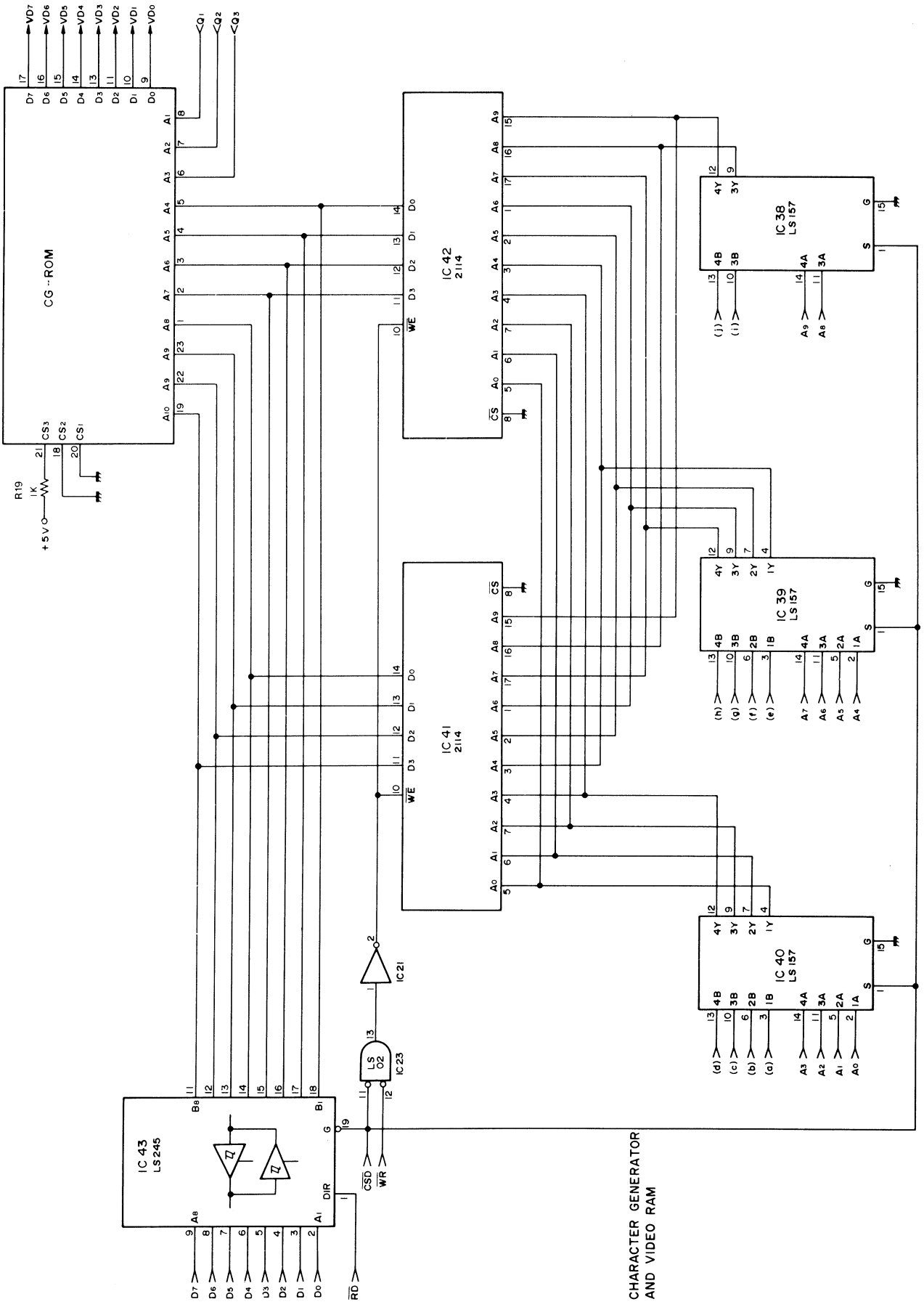
■ CPU Circuit Diagram (3)



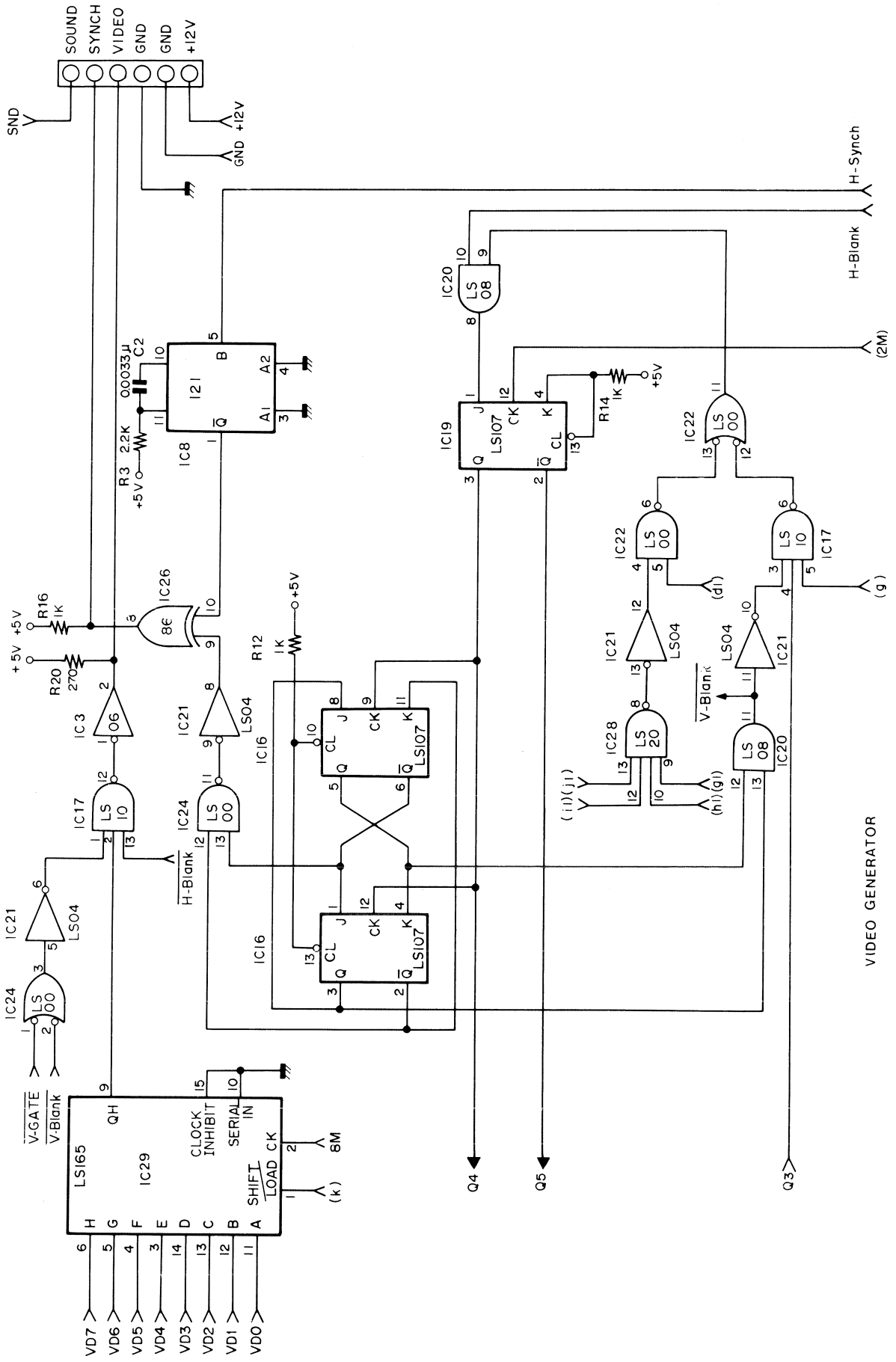
■ CPU Circuit Diagram (4)



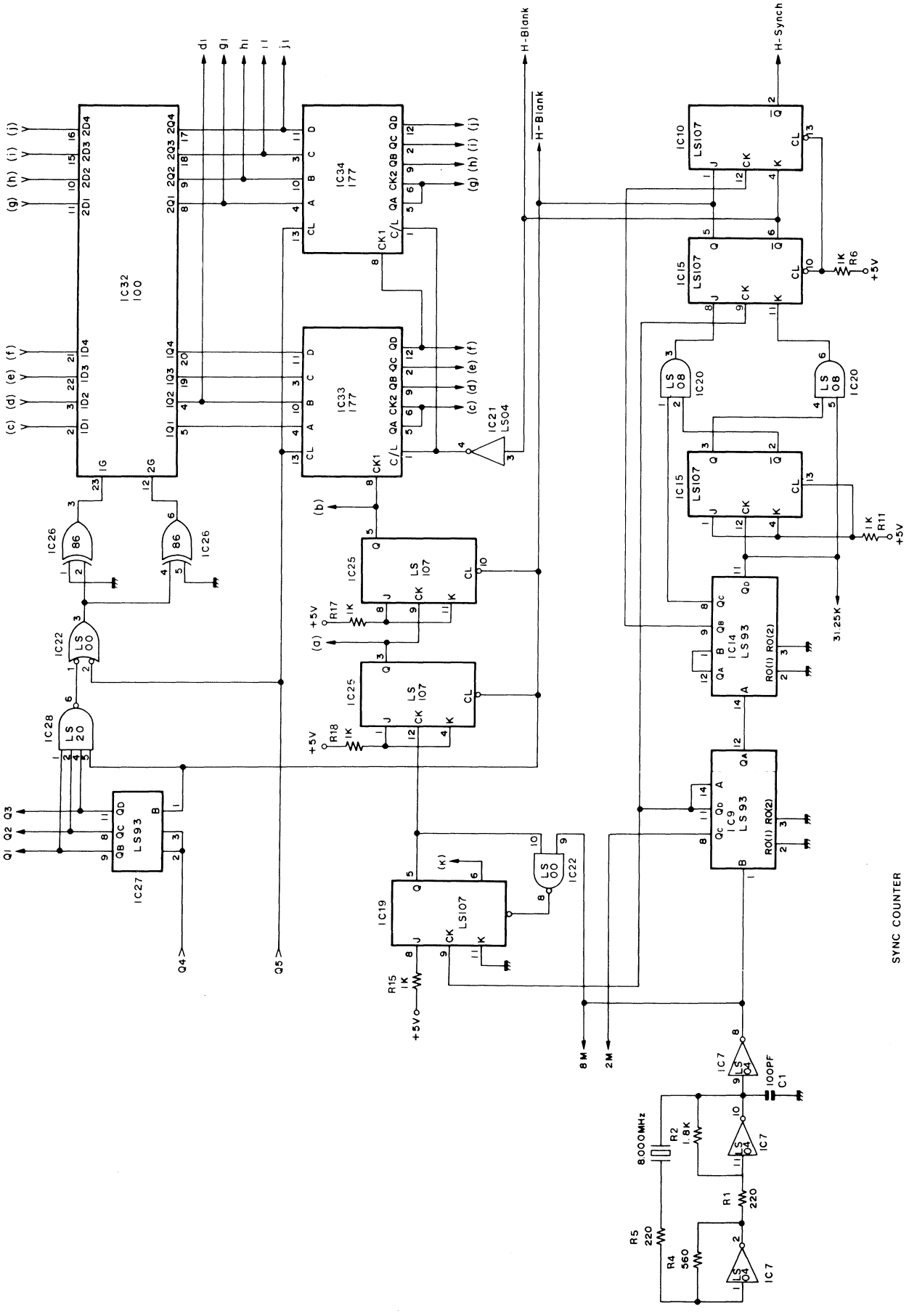
■ CPU Circuit Diagram (5)



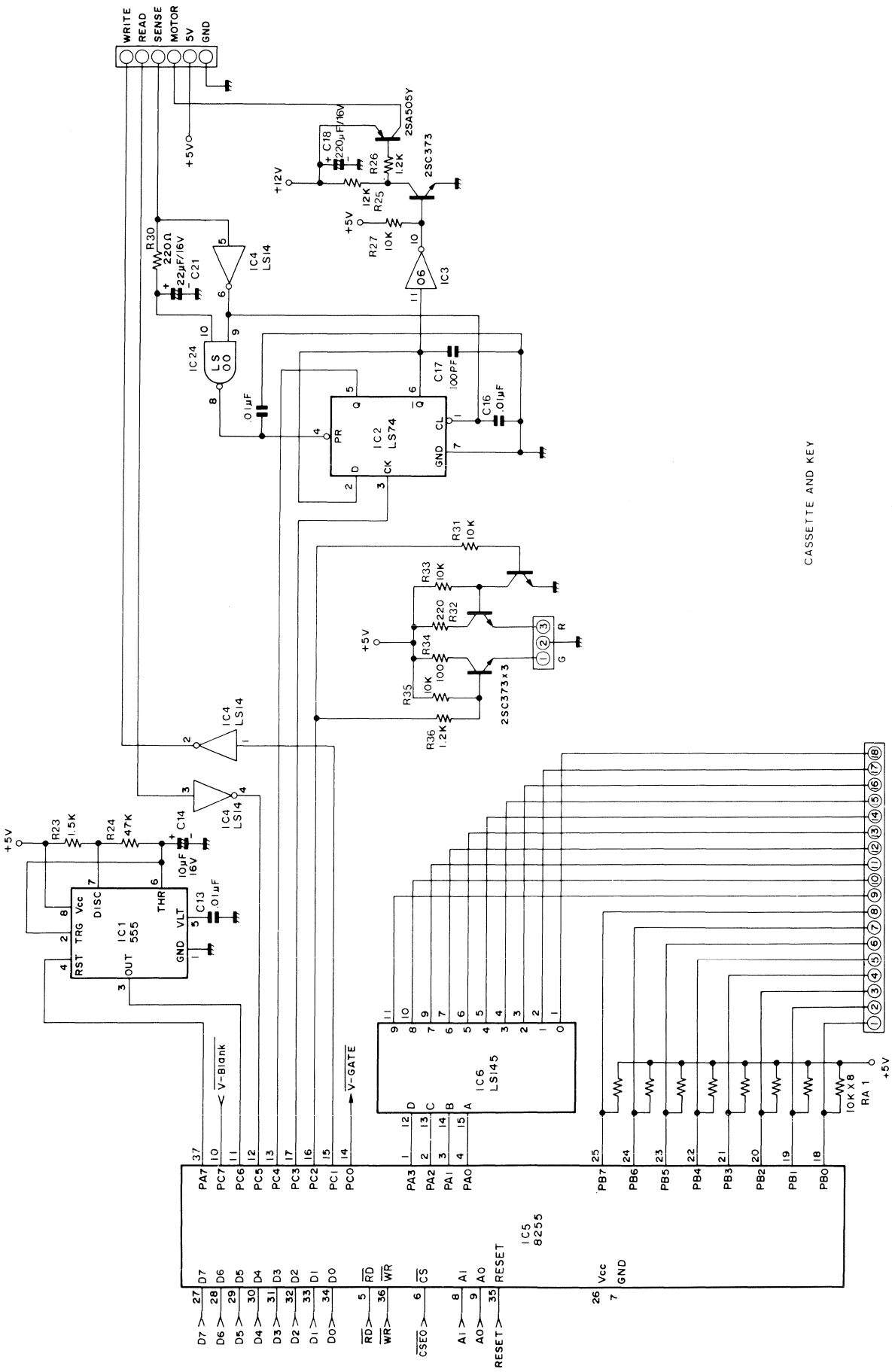
■ CPU Circuit Diagram (6)



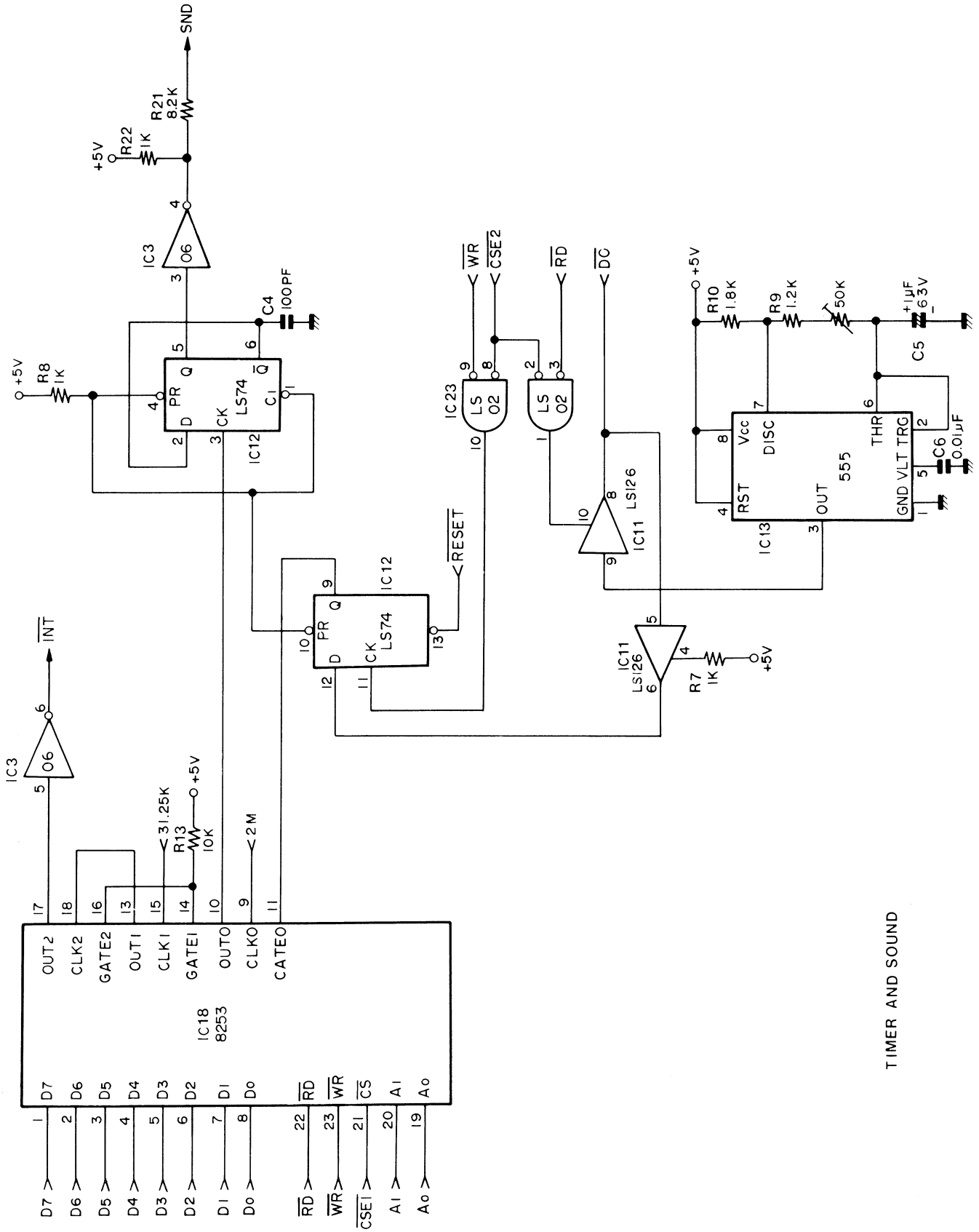
■ CPU Circuit Diagram (7)



■ CPU Circuit Diagram (8)



■ CPU Circuit Diagram (9)

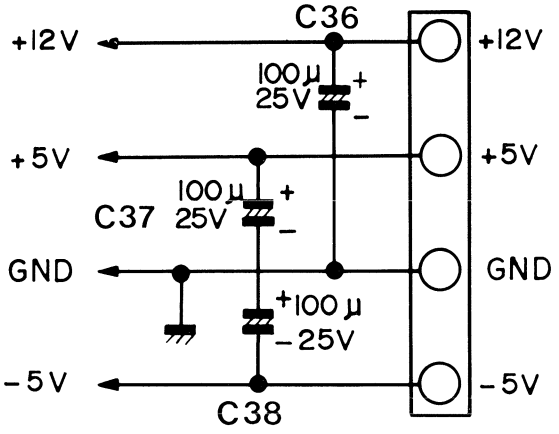


■ CPU Circuit Diagram (10)

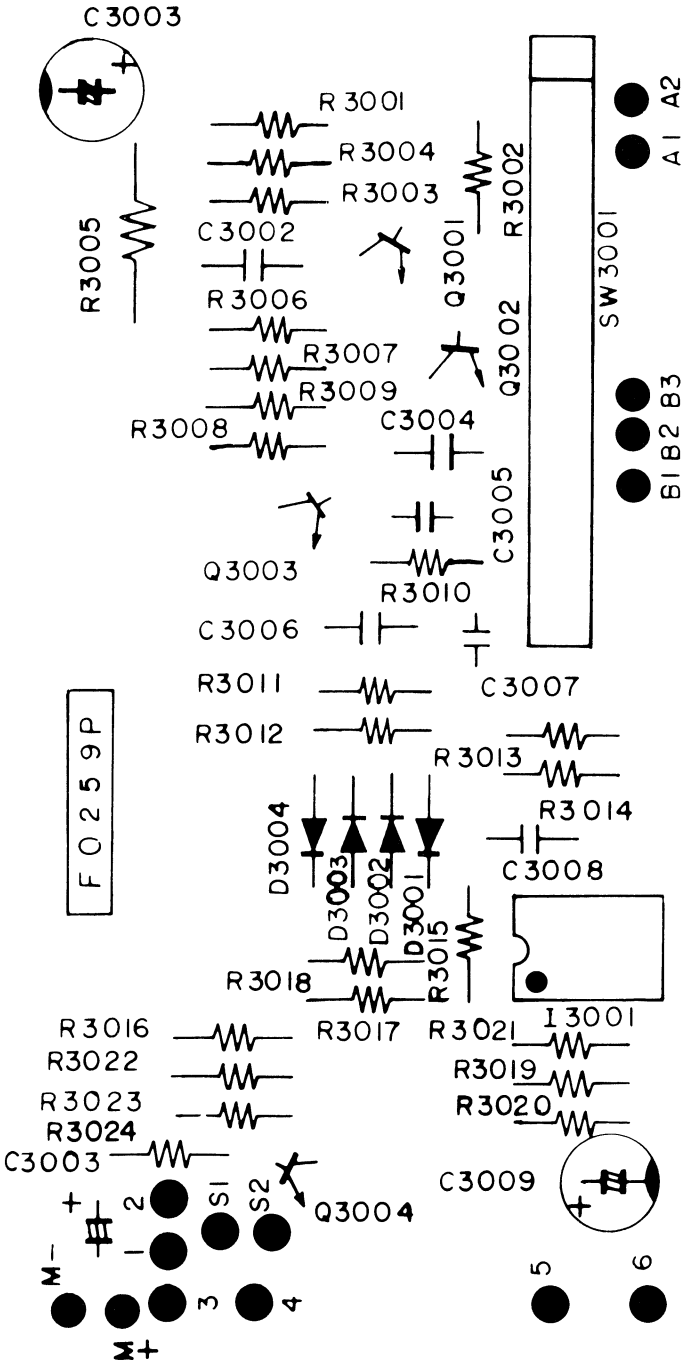
A		B
A15	1	G
A14	2	$\overline{INT}$
A13	3	G
A12	4	$\overline{MREQ}$
A11	5	G
A10	6	$\overline{IORQ}$
A9	7	G
A8	8	$\overline{RD}$
A7	9	G
A6	10	$\overline{WR}$
A5	11	G
A4	12	$\overline{M1}$
A3	13	G
A2	14	$\overline{HALT}$
A1	15	G
A0	16	RESET
G	17	G
D7	18	G
D6	19	G
D5	20	G
D4	21	G
D3	22	G
D2	23	G
D1	24	G
D0	25	G

▷ (MARK)

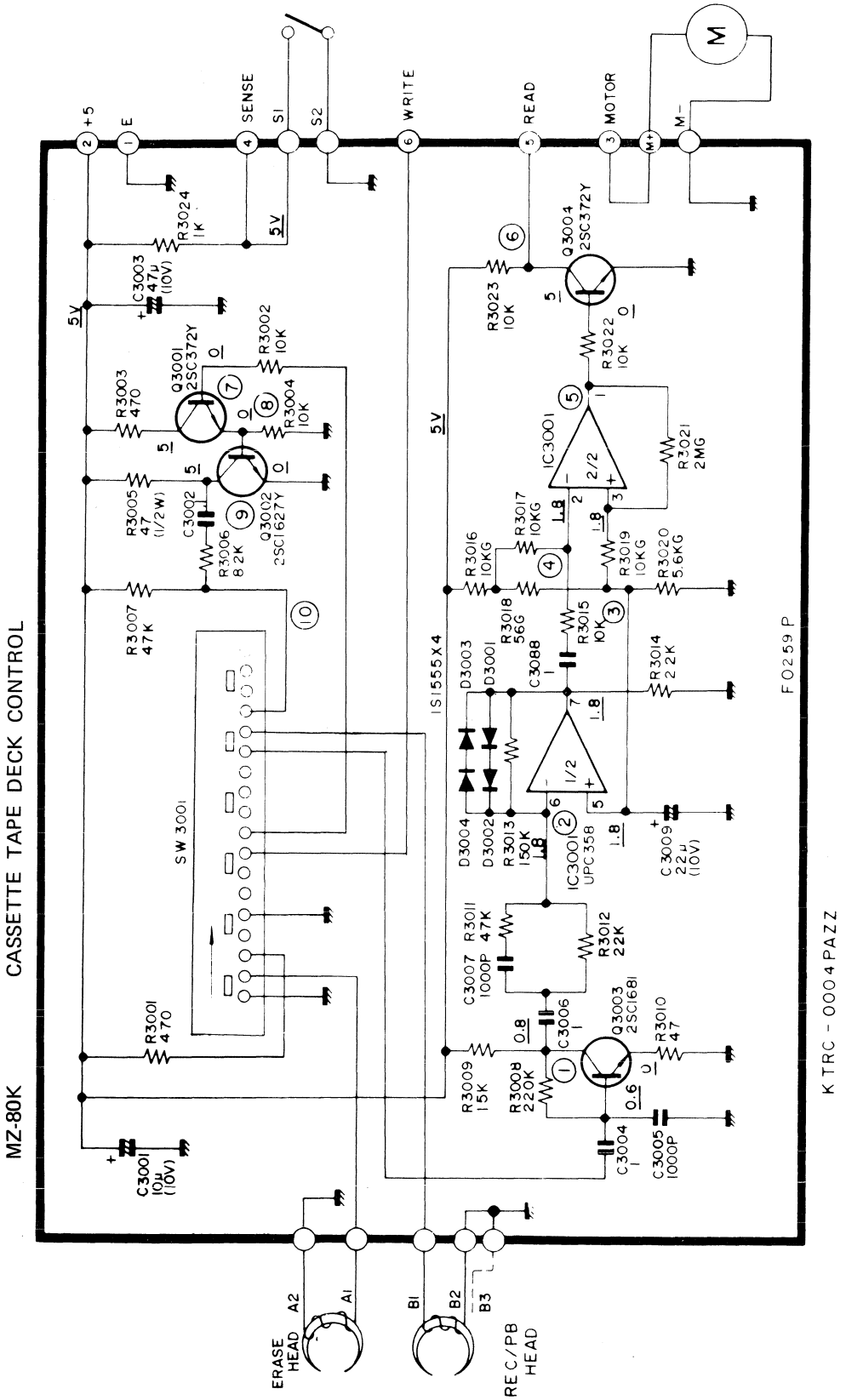
BUS CONNECTOR DETAIL



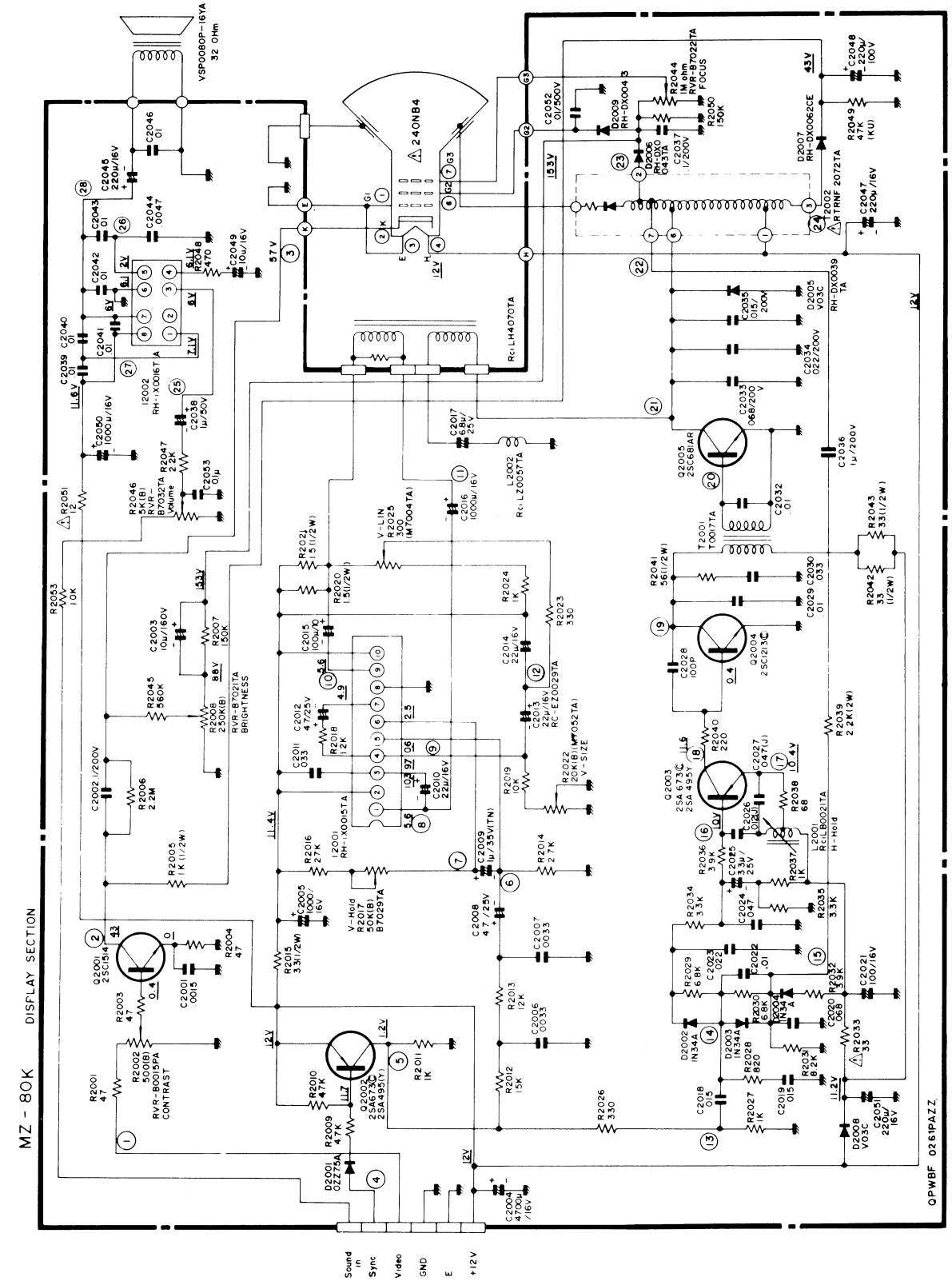
■ Cassette Deck Control Board Component Locations



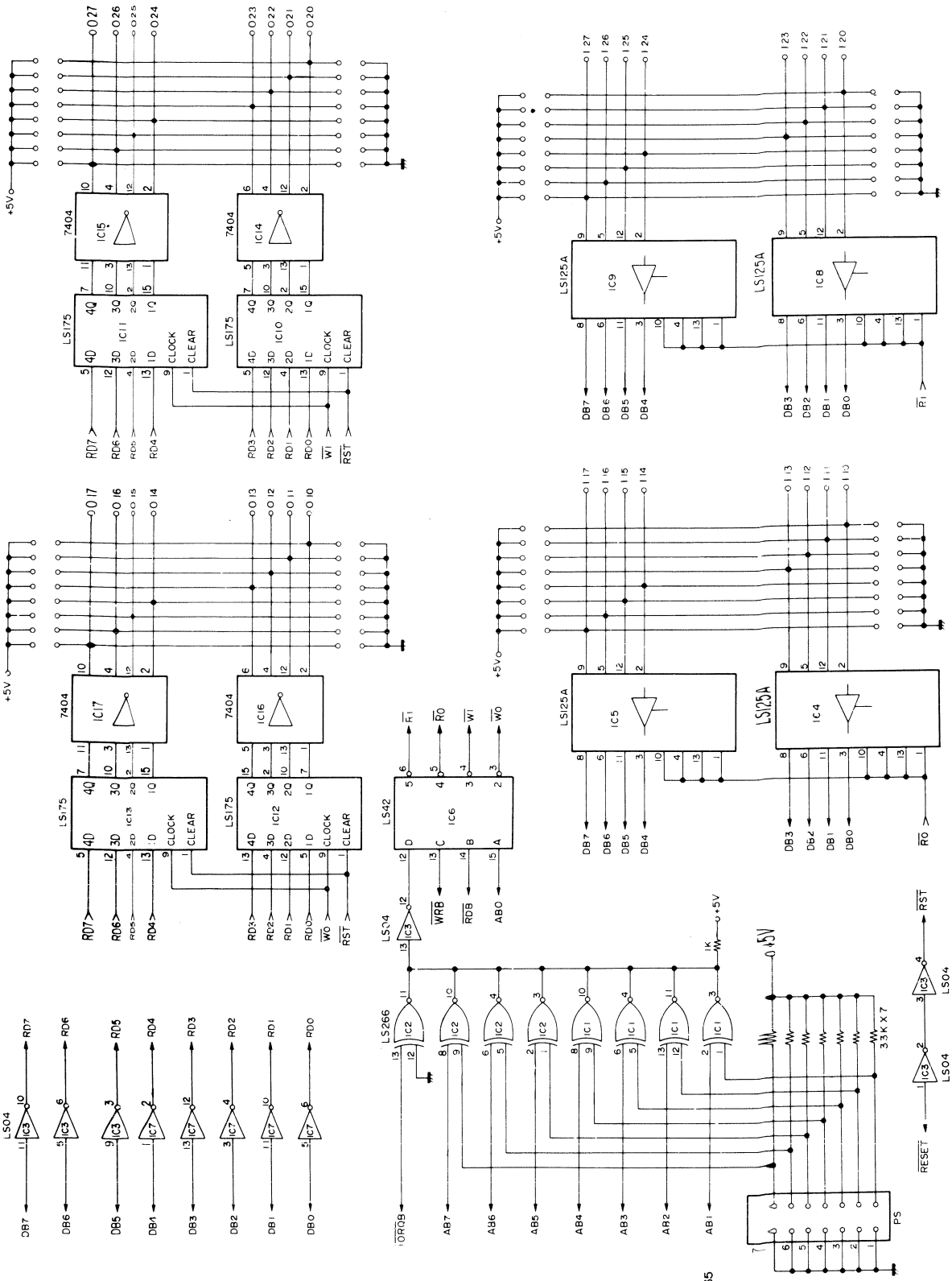
■ Cassette Deck Control Circuit Diagram



■ CRT Display Control Circuit Diagram



UNIVERSAL I/O CARD CIRCUIT DIAGRAM



A		B
	1	DB7
	2	DB6
	3	DB5
	4	DB4
	5	DB3
	6	DB2
	7	DB1
	8	DB0
	9	GND
	10	AB15
	11	AB14
	12	AB13
	13	AB12
	14	AB11
GND	15	AB10
RESET	16	AB9
MBUS	17	AB8
INTB	18	AB7
GND	19	AB6
HALTB	20	AB5
MIB	21	AB4
WRB	22	AB3
RDB	23	AB2
IORQB	24	AB1
MREQB	25	ABO
	26	
+5V	27	+5V
+5V	28	+5V
GND	29	GND
GND	30	GND

A PART'S SIDE

MOTHER BOARD
I/O CONNECTOR
KEL 4800-060-035

ASCII CODE TABLE

ASCII codes used are listed in the following table. Codes 11H through 16H are cursor control codes. For example, executing CALL PRNT with 15H stored in ACC does not display "H" but moves the cursor to the home position.

ASCII

LSD \ MSD	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000			SP	0	@	P			SP		q	n	SP			
1 0001		↓	!	1	A	Q	H				a				♠	●
2 0010		↑	"	2	B	R	I			e	z	Ü				
3 0011		→	#	3	C	S					w	m				♥
4 0100		←	$	4	D	T					s					
5 0101		H	%	5	E	U					u					
6 0110		C	&	6	F	V				t	i		→			X
7 0111			'	7	G	W	●			g		o				o
8 1000			(	8	H	X	☺			h	Ö	l	■			♣
9 1001			)	9	I	Y				k	Ä					
A 1010			*	:	J	Z				b	f	ö				◆
B 1011			+	;	K	[		o		x	v	ä				£
C 1100			,	<	L	\				d			O			↓
D 1101	CR		-	=	M	]				r	ü	y				
E 1110			.	>	N	↑				p	ß	¥				
F 1111			/	?	O	←				c	j					π

DISPLAY CODE TABLE

Display codes are used to call character patterns stored in the character generator. To display a character on the CRT screen, the corresponding display code is transferred to video RAM.

Monitor subroutine PRNT (0012H) or MSG (0015H) converts an ASCII code into the corresponding display code and transfers it to the video RAM location indicated by the cursor. Codes C1H through C6H are the cursor control codes.

DISPLAY