

Personal Computer

117-8000

SYSTEM PROGRAM MANUAL

(EDITOR/ASSEMBLER
SYMBOLIC DEBUGGER)



Personal Computer
mz-800

SYSTEM PROGRAM MANUAL

(EDITOR/ASSEMBLER)
(SYMBOLIC DEBUGGER)

—NOTICE—

This manual is based on the system program (MZ-5Z011 and MZ-1Z018) for the MZ-800/700 personal computers. It describes the MZ-800's editor-assembler (5Z-011A, 1Z-018A) and symbolic debugger (5Z-011B, 1Z-018B) and the MZ-700's editor-assembler (5Z-011C, 1Z-018C) and symbolic debugger (5Z-011D, 1Z-018D).

- (1) The system program for the MZ-800/700 is provided and distributed in files in software packs (cassette tapes or disks).

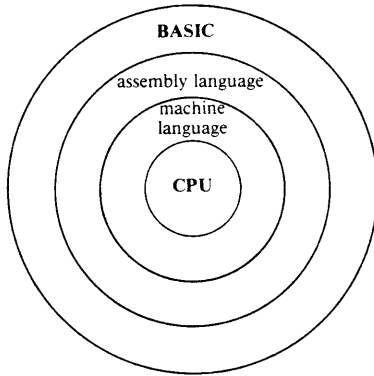
The system program and the contents of this manual are subject to revision without prior notice. Therefore, you are requested to pay special attention to the file version numbers.

- (2) This manual has been carefully prepared and checked for completeness, accuracy and clarity. However, in the event that you should encounter any errors or ambiguities, please feel free to contact your local SHARP representative for clarification.

- (3) The system program for the MZ-800/700 is original product of the SHARP Corporation and all right are reserved. No part of the program and this manual shall be reproduced without permission of the SHARP Corporation.

PREFACE

This manual describes the system program which assists in preparation of assembly language programs for the MZ-800/700 personal computer. Computer programming languages can be classified hierarchically from the lowest level of machine languages to higher levels according to the degree at which they are associated with the hardware of a CPU (Central Processing Unit). For example, the BASIC, assembly language and machine language which can be used to write programs for MZ-800/700 are ranked as illustrated in the figure below.



BASIC, a high level language, allows you to write programs in a notation with which you are far more familiar than assembly and machine languages. Further, you can execute BASIC programs without translating them into machine language programs because the BASIC interpreter translates the BASIC statements of programs into machine codes one by one and executes them immediately. However, you cannot control the CPU operation directly with BASIC as with the machine language and processing speed of BASIC programs is comparatively slow.

On the other hand, the machine language allows you to control the CPU operation directly. Processing speed of machine language programs is fast. However, it is very cumbersome to write programs in the machine language (binary machine codes).

The assembly language described in this manual provides a improved method of writing machine language programs. This manual assumes that the readers are familiar with the contents of the manual provided with the computer. Please refer to it as necessary.

CONTENTS

CHAPTER 1 OUTLINE OF SYSTEM PROGRAM	1
1.1 SYSTEM PROGRAM CONFIGURATION	2
1.1.1 Functions of the text editor	3
1.1.2 Functions of the assembler	3
1.1.3 Functions of the symbolic debugger	4
1.1.4 Object program development procedure using system program	5
1.2 NOTES ON USE OF THE SYSTEM PROGRAM	6
1.2.1 File name and function assignment to function keys	6
1.2.2 Utilities and monitors	7
1.2.3 Other notes	8
CHAPTER 2 EDITOR-ASSEMBLER	9
2.1 OUTLINE OF THE EDITOR-ASSEMBLER	10
2.2 TEXT EDITOR	11
2.2.1 Outline of the text editor.....	11
2.2.2 Character pointer and delimiter.....	13
2.2.3 Text editor commands.....	15
\ DEFAULT command	15
\ DIR command	15
\ DIR/P command.....	15
\ INIT command	16
\ MODE command	16
\ RUN command	16
\ DELETE command	17
\ RENAME command	17
\ LOADALL command	18
\ SAVEALL command	18
R (Read file) command	18
A (Append file) command	19
W (Write) command	20
V (Verify) command	21
T (Type) command	22
B (Begin) command	23
Z command	23
J (Jump) command	23
L (Line) command.....	24
M (Move) command	24
C (Change) command.....	25
Q (Queue) command	25

I (Insert) command	26
K (Kill) command	27
D (Delete) command	28
S (Search) command	29
= (equal) command	30
. (period) command	30
& (ampersand) command	30
X (TRANSfer) command	30
# (sharp mark) command	31
! (exclamation mark) command	31
2.3 ASSEMBLER	32
2.3.1 Outline of the assembler	32
2.3.2 Assembly language rules	35
2.3.3 Assembly listing and assembler messages	40
2.3.4 Assembler directives	43
ENT (ENTry)	43
EQU (EQUate)	44
ORG (ORiGin)	45
MACRO/ENDM	46
IF, IFF, IFT, IFD, IFU/ENDIF	47
DEFBn (DEFine Byte)	49
DEFB'S', DEFB''S'' (DEFine Byte)	49
DEFWnn' (DEFine Word)	50
DEFM'S', DEFM''S'' (DEFine Message)	50
DEFSnn' (DEFine Storage)	51
LIST, UNLIST	52
SKPn (SKiP n lines)	53
SKPH (SKiP Home)	53
END (end)	53
2.4 ERROR MESSAGES	54
2.4.1 Monitor error messages	54
2.4.2 Text editor error messages	54
2.4.3 Assembler error messages	55
 CHAPTER 3 SYMBOLIC DEBUGGER	 57
3.1 OUTLINE OF THE SYMBOLIC DEBUGGER	58
3.2 BREAKPOINTS	60
3.3 SYMBOL TABLE	61
3.4 LINKER BIAS AND ADDRESSES	62
3.5 SYMBOLIC DEBUGGER COMMANDS	66
L (relocatable Load) command	66
N (Next file) command	67
H (Height) command	67

= (table dump) command	68
* (clear bias and table) command	68
B (Breakpoint) command	70
& (clear breakpoint) command	71
T (Trace) command	72
M (Memory dump) command	73
D (Disassemble) command	74
W (data Write) command	76
G (Goto) command	77
? (search) command	78
F (Fill memory) command	79
A (Accumulator) command	80
C (Complementary) command	80
P (Program counter) command	81
R (Register) command	81
X (data TRANSfer) command	83
\ DEFAULT command	84
\ DIR command	84
\ DIR/P command	84
\ INIT command	85
\ MODE command	85
\ RUN command	85
\ DELETE command	86
\ RENAME command	86
\ LOADALL command	86
\ SAVEALL command	86
S (Save) command	87
Y (Yank) command	88
V (Verify) command	89
I (Inport) command	90
O (Outport) command	90
# (sharp mark) command	91
! (exclamation mark) command	91

3.6 ERROR MESSAGES	92
3.6.1 Monitor error messages	92
3.6.2 Symbolic debugger error messages	93

CHAPTER 4 SAMPLE PROGRAMS	95
4.1 DRAWING AN APPROACHING SQUARES	96
4.2 DISPLAYING BINARY DATA IN HEXADECIMAL REPRESENTATION	99
4.3 ENTERING HEXADECIMAL DATA	101
4.4 DISPLAYING A MEMORY BLOCK	104
4.5 WRITING DATA INTO A MEMORY AREA	107

Appendices	111
1. MONITOR SUBROUTINES	112
MZ-800 monitor subroutines	112
MZ-800 monitor call	115
Examples of use of monitor calls.....	123
2. MAKING BACKUP COPY OF THE MZ-800 SYSTEM PROGRAM	127
COPYING MZ-700 SYSTEM PROGRAM	128
3. CODE TABLES	130



CHAPTER 1

**OUTLINE OF SYSTEM
PROGRAM**

1.1 SYSTEM PROGRAM CONFIGURATION

This system program consists of an editor-assembler and a symbolic debugger. The editor-assembler and symbolic debugger execute a job at each stage of the assembly process, respectively.

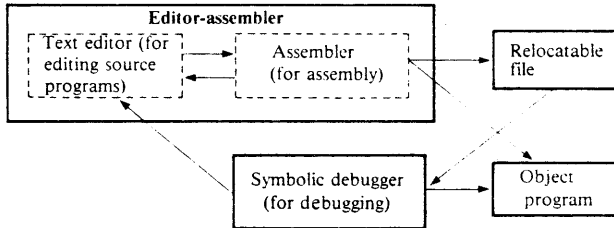


Fig. 1-1 Outline of the assembly process

Figure 1-1 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 above figure makes up a program creation stage. The programmer prepares a source program with the text editor and creates a source file, then inputs it to the assembler. The assembler analyzes and interprets the syntax of the source program according to the assembly language rules, and assembles the source program into relocatable binary code. When the assembler detects errors, it issues error messages. The programmer then corrects the errors in the source program with the text editor.

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

After creating a complete source program and assembling it by the procedure above, an object program can be obtained by loading the program in relocatable form output by the editor-assembler using the symbolic debugger. When a program is to be made up of two or more program units, those program units must be loaded by being relocated and then linked each other. An object program can also be obtained directly with the editor-assembler by specifying the assembly option.

1.1.1 Functions of the text editor

The primary functions of the text editor include those used for making insertion, deletion and change in source programs. The text editor displays the source program on the CRT screen and allows you to edit it interactively. This makes it possible to perform these tasks with a minimum of effort.

The command format employed for this text editor is compatible with that of the editor program of the NOVA minicomputer manufactured by the Data General Corporation and perfected over a period of many years by many users. The figure below shows the general flow of the process to edit a source program with the text editor.

- ① A source program is loaded from a disk or other storage device into the memory.
- ② The source program is displayed on the CRT screen and modified by making insertions, deletions and changes.
- ③ After all modifications have been made, the source program is written onto a disk or other storage device again, or assembled by the assembler.

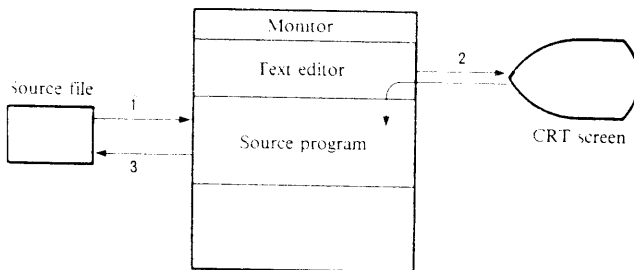


Fig. 1-2 General flow of procedure to edit a source program using text editor

1.1.2 Functions of the assembler

The assembler converts programs written in assembly language into machine language. In other words, source programs composed of ASCII code which are prepared using the text editor are read and used to prepare relocatable programs composed of arrays of binary numbers. This process can be broadly divided into four steps, as follows.

- (1) Identifying label symbols and storing them in a symbol table.
- (2) Identifying mnemonics and generating their object codes.
- (3) Preparing assembly lists.
- (4) Preparing relocatable files.

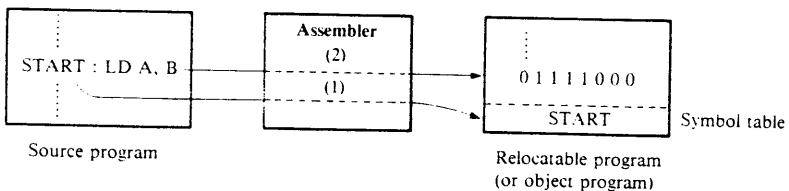


Fig. 1-3 Functions of the assembler

1.1.3 Functions of the symbolic debugger

The symbolic debugger loads a machine language program in relocatable form output by the assembler into the link area of the memory, converts it into the format in which it can be executed, then outputs it (object program).

The symbolic debugger links two or more relocatable program units to produce a single object program. That is, machine language programs output by the assembler are organized so that they can be loaded in any memory areas and converted into the form immediately executable in the memory areas in which they are loaded. When two or more relocatable program units are to be linked, they are loaded in memory areas which do not overlap each other and reorganized so that they can be executed in those memory area. This is the relocate function of the symbolic debugger.

In many cases, some of those programs to be linked reference symbols defined in other programs. When the symbolic debugger links them, it ensures that the external references are made properly. This is the link function of the symbolic debugger. **Fig. 1-4** below shows linkage of two programs with the symbolic debugger.

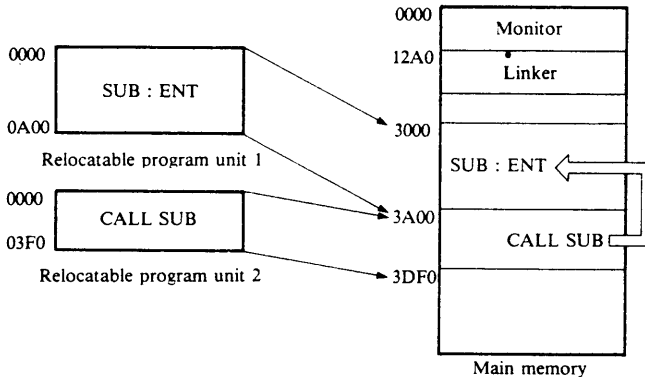
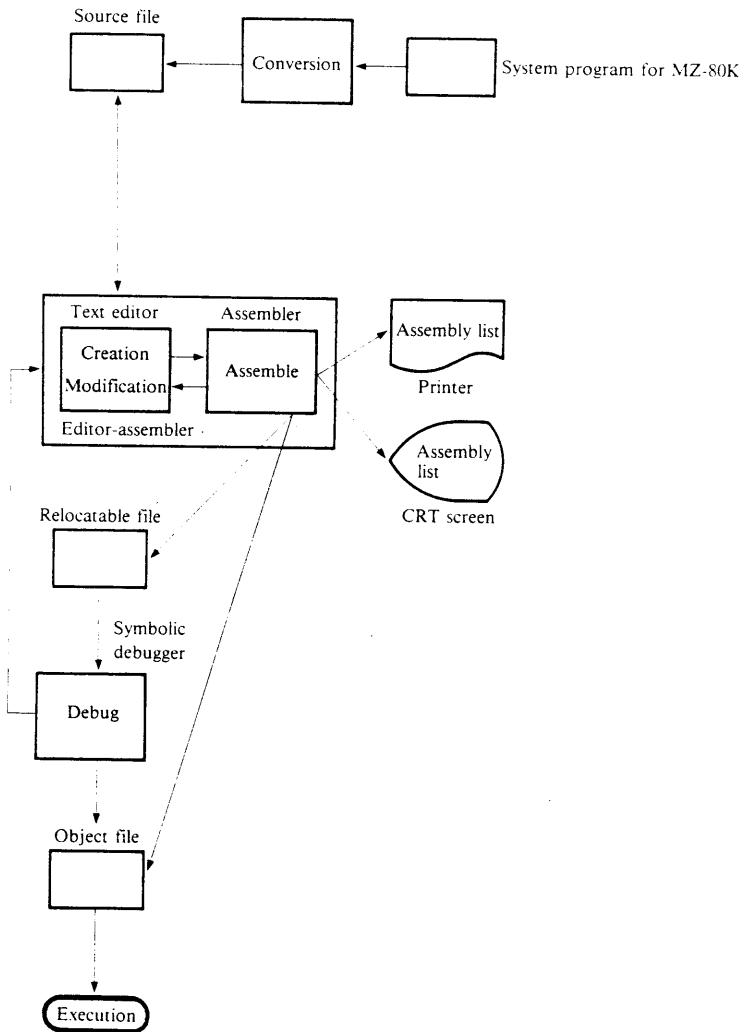


Fig. 1-4 Functions of the linker

The symbolic debugger allows to debug an object program in executable form in the link area by actually executing it.

Debugging is performed using break points. That is, program execution stops at each break point set in the program to allow the status of the system to be checked. The symbolic debugger also provides the function to set breakpoints in the program.

1.1.4 Object program development procedure using system program



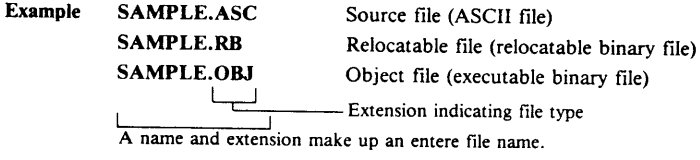
1.2 NOTES ON USE OF THE SYSTEM PROGRAM

1.2.1 File name and function assignment to function keys

When you store programs and data on external storage media (disk, cassette tape and so on), you must assign a file name to them. Files stored on external storage media can be read from the external storage media and loaded in the memory of the computer using the file names assigned to them when they were stored.

This system program allows to assign any name made up of up to 16 characters (letters and symbols). However, if you intend to use other system programs too, use of names made up of only alphabetic and numeric characters is recommended, because those system programs may not be able to read file names including symbols.

It is not possible to store two or more files on a single disk under the same name even if their file types are different. To assign the same name to two or more different types of files and to store those files on a single disk, add an extension which indicates the file type after each file name.



The functions assigned to the function keys by the editor-assembler and the symbolic debugger are as follows. The functions assigned to the function keys cannot be changed by the user.

	Editor-assembler	Symbolic debugger
F1	" \ RUN_"	" RUN_"
F2	" \ DEFAULT_"	" DEFAULT_"
F3	" .ASC"	" .ASC"
F4	" .RB"	" .RB"
F5	" [X]" (used as delimiter)	" DIR_"
SHIFT + F1	" FD1:"	" FD1:"
SHIFT + F2	" FD2:"	" FD2:"
SHIFT + F3	" QD:"	" QD:"
SHIFT + F4	" CMT:"	" CMT:"
SHIFT + F5	" \ DIR_"	" \ DIR / P_"

1.2.2 Utilities and monitors

The following utility programs are included among the programs provided together with the MZ disk BASIC which support MZ disks.

- **DELETE utility**
Deletes unneeded files on MZ disks to allow effective use of MZ disks.
- **QDCOPY utility**
Copies an entire contents of a MZ-disk onto another one.
- **TRANS utility**
Converts the format of files created with the MZ-80K system programs into that in which this system program can write and read those files to/from MZ disks correctly, and writes them on MZ disks.

(Refer to the manual of the MZ disk BASIC for detailed explanations of the above utility programs.)

The above utility programs can be loaded and executed by the editor assembler or symbolic debugger too. For example, the TRANS utility program can be started by the following operation.

* \ RUN "TRANS" **[CR]** (RUN "TRANS" **[CR]** for BASIC)

The DELETE and QDCOPY utility programs can also be loaded and executed by the same operation. After these utility programs are started, operate following the messages displayed on the screen by the program.

Monitor programs used with the BASIC and this system program are as follows.

Monitor A

MZ-700	BASIC	1Z-013A
--------	-------	---------

Monitor B

MZ-700	MZ DISK BASIC	5Z-008
MZ-700	DISK BASIC	2Z-009
MZ-700	Editor-assembler	1Z-018C, 5Z-011C
MZ-700	Symbolic debugger	1Z-018D, 5Z-011D

Monitor C

MZ-800	BASIC	1Z-016
MZ-800	MZ DISK BASIC	5Z-009
MZ-800	DISK BASIC	2Z-046
MZ-800	Editor-assembler	1Z-018A, 5Z-011A
MZ-800	Symbolic debugger	1Z-018B, 5Z-011B

Monitor D

MZ-80K	BASIC
MZ-80K	Double precision
MZ-80K	FDOS
MZ-80K	System program

1.2.3 Other notes

- Lowercase letters cannot be typed in while the system program (editor-assembler and symbolic debugger) is in the command wait state and while an assembly language program is entered under the editor-assembler.
- \LOADALL, \SAVEALL, \DELETE, and \RENAME commands among the file control commands of this system program cannot be used if the optional RAM file (MZ-1R18) is not installed.
- In the MZ-700 mode, RAM file related commands of both the editor-assembler and the symbolic debugger cannot be used.