Personal Computer

MZ-800

SYSTEM PROGRAM MANUAL

(EDITOR/ASSEMBLER
SYMBOLIC DEBUGGER)

Personal Computer

MZ-800

SYSTEM PROGRAM MANUAL

(EDITOR/ASSEMBLER
SYMBOLIC DEBUGGER)

NOTICE

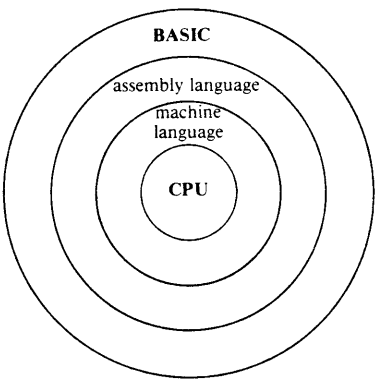
This manual is based on the system program (MZ-5Z011 and MZ-1Z018) for the MZ-800/700 personal computers. It descirbes the MZ-800's editor-assembler (5Z-011A, 1Z-018A) and symbolic debugger (5Z-011B, 1Z-018B) and the MZ-700's editor-assembler (5Z-011C, 1Z-018C) and symbolic debugger (5Z-011D, 1Z-018D).

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

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 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.

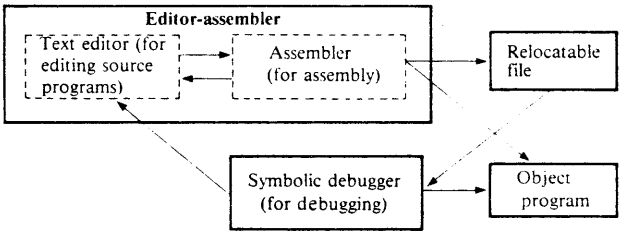


Fig. 1-1 Outline of the assembly process

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 reedits 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.

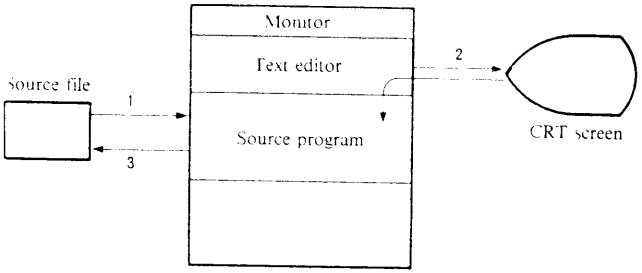


Fig. 1-2 General flow of procedure to edit a soruce program using text editor

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.

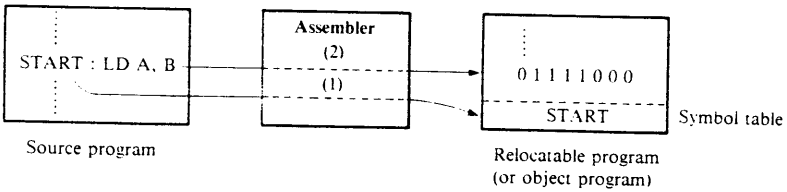


Fig. 1-3 Functions of the assembler

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.

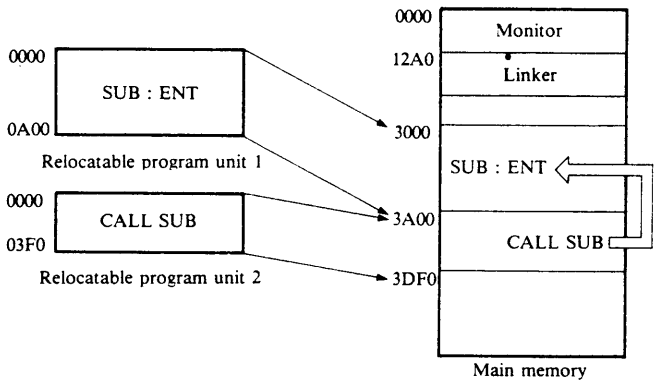
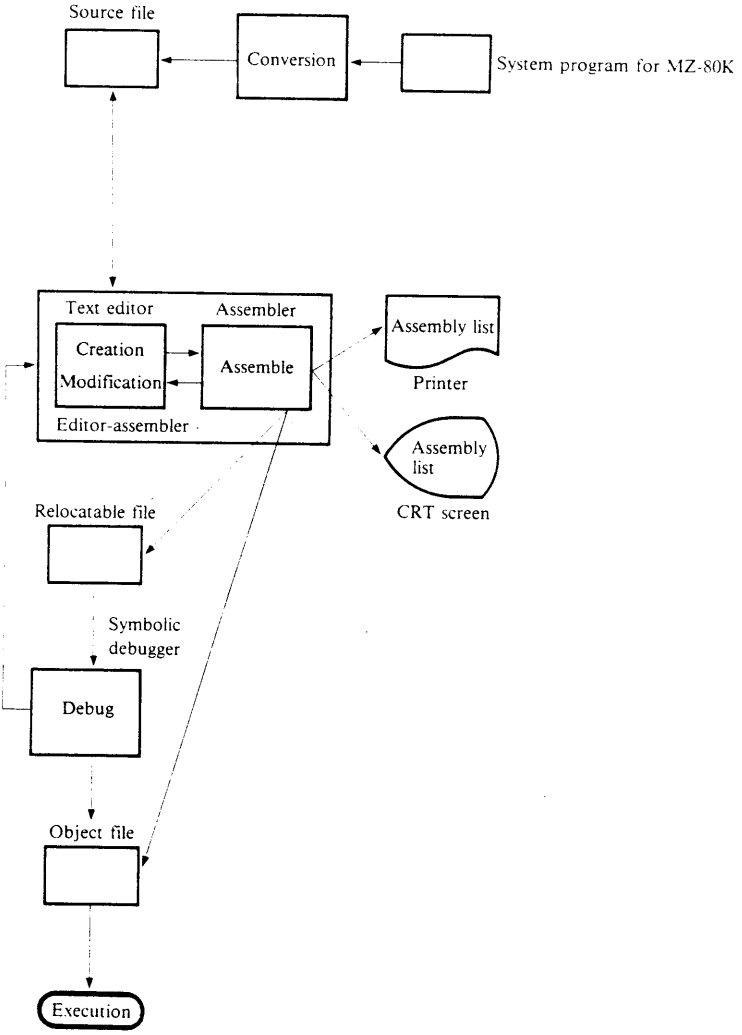


Fig. 1-4 Functions of the linker

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



1.2.1 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 assinged 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.

Example

SAMPLE.ASC	Source file (ASCII file)
SAMPLE.RB	Relocatable file (relocatable binary file)
SAMPLE.OBJ	Object file (executable binary file)

Extension indicating file type

A name and extension make up an entere 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	" \ R U N␣ "	" R U N␣ "
F2	" \ D E F A U L T␣ "	" D E F A U L T␣ "
F3	" . A S C "	" . A S C "
F4	" . R B "	" . R B "
F5	" ▩ " (used as delimitor)	" D I R␣ "
SHIFT + F1	" F D 1 : "	" F D 1 : "
SHIFT + F2	" F D 2 : "	" F D 2 : "
SHIFT + F3	" Q D : "	" Q D : "
SHIFT + F4	" C M T : "	" C M T : "
SHIFT + F5	" \ D I R␣ "	" \ D I R / 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 diks.

- 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

Moinitor 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.