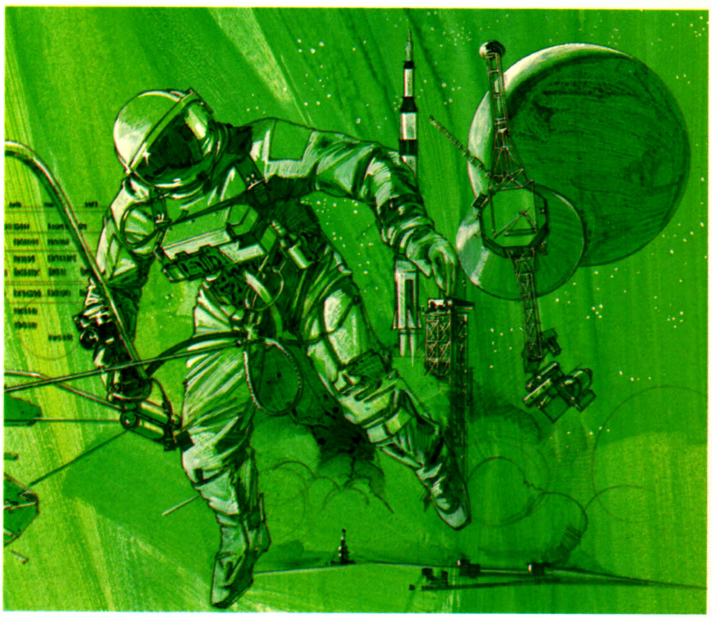
Personal Computer

MZ-800

OWNER'S MANUAL





SHARP®

Personal Computer

MZ-800

Owner's Manual

© SHARP CORPORATION

NOTICE

This manual has been written for the MZ-800 personal computers and the BASIC interpreter (1Z-016) which is provided with the MZ-800.

(1) All system software for the MZ-800 computers is supported in software packs (cassette tape, etc.) in file form. The contents of all system software and the material presented in this manual are subject to change without prior notice for the purpose of product improvement and other reasons, and care should be taken to confirm that the file version number of the system software used matches that specified in this manual.

(2) All system software for the Sharp MZ-800 personal computer has been developed by the Sharp corporation, and all rights to such software are reserved. Reproduction of the system software or the contents of this book is prohibited.

(3) This computer and the contents of this manual have been fully checked for completeness and correctness prior to shipment; however, if you should encounter any problems during operation or have any questions which cannot be resolved by reading this manual, please do not hesitate to contact your Sharp dealer for assistance.
Not withstanding the foregoing, note that the Sharp Corporation and its representatives will not assume responsibility for any losses or damages incurred as a result of operation or use of this equipment.

Preface

Congratulations on purchasing the MZ-800 computer. Your MZ-800 is a compact personal computer, featuring 640 × 200 dot addressable graphics, 16-colour display, and a programmable sound generator (PSG) which can generate 3-tone chords over 6 octaves. One of the excellent features of the MZ-800 is that it contains hardware which makes it compatible with the MZ-700 series computer. This makes it possible for you to use most of the existing programs for the SHARP MZ-700 series computers on your MZ-800.

This manual is written both as a guide to the MZ-800 and a BASIC reference manual. The manual is constructed as follows.

Chapter 1 describes how to unpack, handle, and setup your MZ-800, and what to do if a problem occurs.

Chapter 2 describes how to turn on the power, load the BASIC interpreter, and turn off the power.

Chapter 3 explains the BASIC interpreter. This chapter also shows you how to write a simple program, edit it, save it on a cassette tape and load it back into memory.

Chapter 4 describes the functions of the keyboard keys. This chapter also describes how to operate the data recorder and handle tapes.

Chapter 5 presents the background knowledge you need to be able to write programs.

Chapter 6 describes the BASIC commands and statements.

Chapter 7 describes the hardware configuration of the MZ-800 and I/O port control. It also describes peripheral devices and how to connect them.

Chapter 8 explains the monitor program, which allows you to "bypass" BASIC and directly access the MZ-800's memory.

Chapter 9 explains the MZ-700 mode of the MZ-800.

Make sure that you read the handling and setup instructions before turning on the computer's power switch. Read this manual thoroughly to get the most out of your MZ-800 computer.

CONTENTS

Chapter 1 Introduction
1.1 Unpacking 1-2
1.2 Handling 1-3
1.3 Appearance 1-4
1.4 Setup 1-6
1.5 In Case of Difficulty 1-10

Chapter 2 Start Up
2.1 Power-on 2-2
2.2 Power-off 2-3
2.3 Running the Demonstration Program 2-4

Chapter 3 Basic Operation
3.1 Introduction 3-2
3.2 Getting to know the Keyboard 3-3
3.3 Writing a Simple Program 3-4
3.4 Editing Programs 3-7
3.5 Saving a Program 3-10
3.6 Loading a Program 3-12

Chapter 4 Keyboard and Data Recorder
4.1 Keyboard 4-2
4.2 Data Recorder 4-6

Chapter 5 Programming Concepts
5.1 Multi-statement Lines and Line Numbers 5-2
5.2 Numeric Data and String Data 5-3
5.3 Constants 5-4
5.4 Variables 5-5
5.5 Array Variables 5-6
5.6 Expressions 5-7
5.7 Files 5-9
5.8 Functions 5-10
5.9 Screen Coordinates 5-16

Chapter 6 MZ-800 BASIC Commands and Statements
6.1 Commands 6-3
6.2 Fundamental Statements 6-9
6.3 File Control Statements 6-43
6.4 Graphics Control Statements 6-56
6.5 Music Control Statements 6-68
6.6 Printer Control Statements 6-73
6.7 Machine Language Control Statements 6-84
6.8 Error Processing Statements 6-87

Chapter 7 Hardware

7.1 MZ-800 Hardware 7-2
7.1.1 System diagram 7-2
7.1.2 System switch settings 7-3
7.1.3 I/O port control 7-4
7.2 Peripheral Devices 7-8
7.2.1 Standard interfaces 7-8
7.2.2 Expansion I/O connector 7-8
7.2.3 RAM file board (MZ-1R18) 7-10
7.2.4 Joystick 7-11
7.2.5 Printers 7-13
7.2.6 Optional graphic memory MZ-1R25 7-16
7.2.7 External cassette tape recorder (for MZ-811 only) 7-18

Chapter 8 Monitor

8.1 General 8-2
8.2 ROM Monitor and BASIC Monitor 8-3
8.3 Starting the ROM Monitor 8-4
8.4 Monitor Commands 8-5
8.5 BASIC Monitor 8-8
8.6 BASIC Monitor Commands 8-9

Chapter 9 MZ-700 Mode

9.1 Using MZ-700 Programs 9-2
9.2 Summary of MZ-700 BASIC Commands and Statements, Functions and Operations 9-3

Appendixes

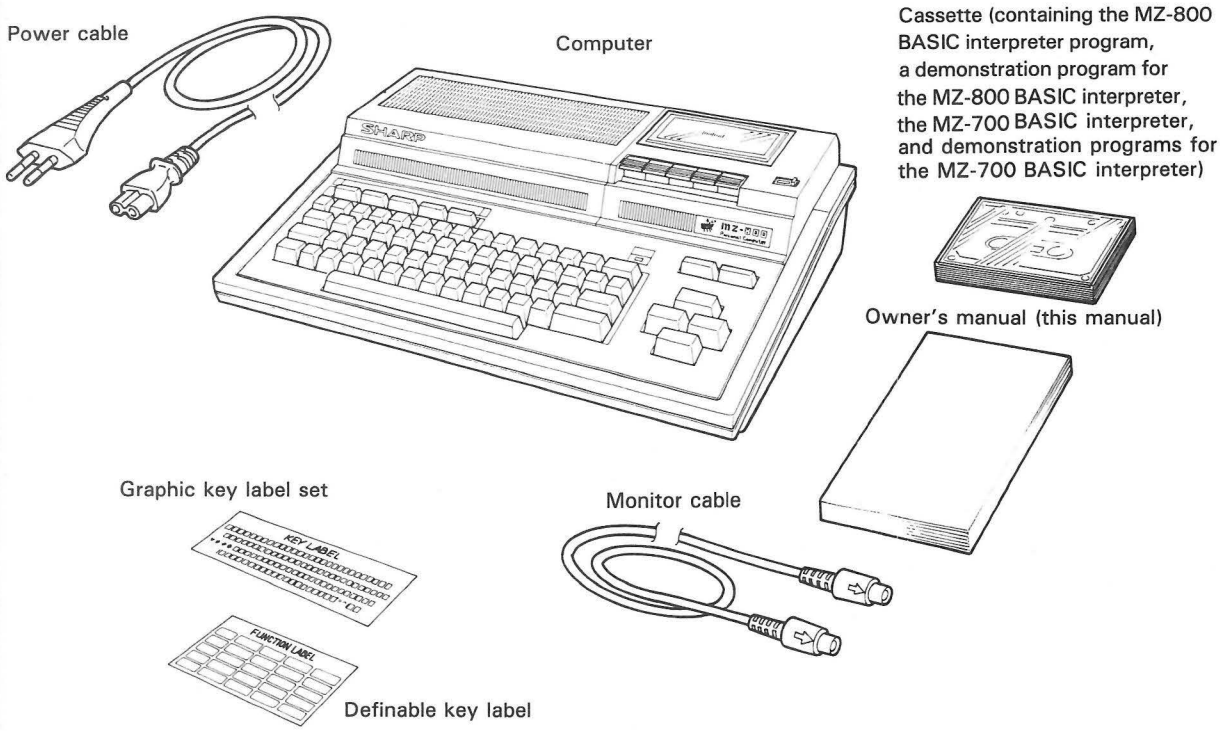
Appendix A Display Control in the MZ-800 Mode A-2
B Programmable Sound Generator A-7
C Reserved Words A-10
D Console Control Codes A-12
E Restrictions on Using File I/O Commands and Statements A-13
F Monitor Subroutines A-14
G Making Backup Copy of the BASIC Interpreter A-17
H Optional Colour Plotter-Printer MZ-1P16 ..... A-18
I Colour Plotter-Printer Control Codes A-21
J Code Tables A-24
K Error Messages Generated by the Monitor A-26
L Error Messages Generated by BASIC A-27
M Index A-29
N Specification A-31

Chapter 1 Introduction

This chapter describes how to handle and set up the MZ-800 computer system. Read this chapter carefully before turning on the power switch.

1.1 Unpacking

Remove the MZ-800 from the packing carton and check that you have the following items.



Store the carton and packing materials away in a safe place, so that you can reuse them if you have to transport the computer in the future.

1.2 Handling

1) This computer uses many precision parts. Do not use or store it in extremely hot or cold conditions, or under conditions where the temperature changes rapidly.
2) Do not use or store the computer in damp or dusty places, and avoid exposing it to corrosive chemicals or gases.
3) Do not block the ventilation holes or place large objects nearby that will disrupt ventilation.
4) Do not subject the computer to shock or vibration.
5) Do not expose the computer to direct sunlight.
6) Do not allow water or other liquid to enter the cabinet. Using the computer when it is wet is very dangerous, and will damage the computer's electronics.
7) Do not disassemble the cabinet unless you are installing options as instructed by documents from SHARP.
8) Radios and TV sets may pick up interference from RF (radio frequency) noise generated by the computer. Keep such equipment (other than that you may be using as the computer's display unit) well away from the computer.
9) When peripheral devices are connected, the display image may jitter. If this problem occurs, change the layout of your system's equipment.
10) Do not place any object other than the optional plotter/printer (MZ-1P16) on the cabinet.
11) After turning off the power switch, unplug the power cable by grasping the plug molding, not the cable.
12) Make sure that you turn off the power switch when you not using the computer. After turning off the power switch, wait at least 10 seconds before turning it on again, otherwise the system may not operate properly.
13) Use a dry soft cloth to clean the unit. Do not use a wet cloth or volatile fluids such as alcohol or benzene. Discolouration or deformation of the cabinet may result if this precaution is ignored.
14) If you notice any abnormal condition such as an extremely high temperature, an abnormal odour, or smoke, stop what you are doing and quickly turn off the power then unplug the power cable.

MZ-811 and MZ-821

One of the models described in this manual may not be available in some countries.

This manual explains two personal computer models: the MZ-811 and the MZ-821. Differences between these two computers are as follows.

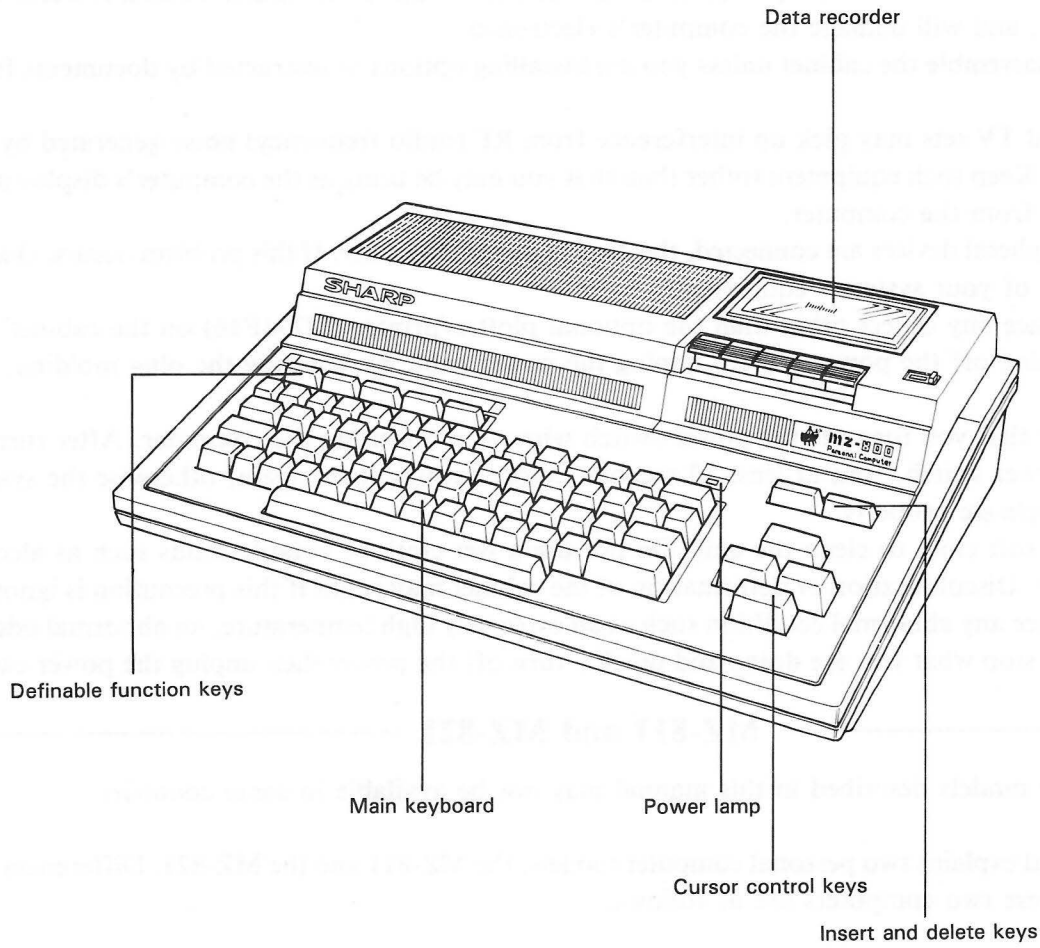
Model name	MZ-811	MZ-821
Data recorder	Optional	Standard
Ordinary cassette recorder	Connectable	Not connectable

When the optional MZ-1T04 data recorder is installed on the MZ-811, it becomes equivalent to the MZ-821. Procedures for installing the data recorder are described in the MZ-1T04 instruction manual.

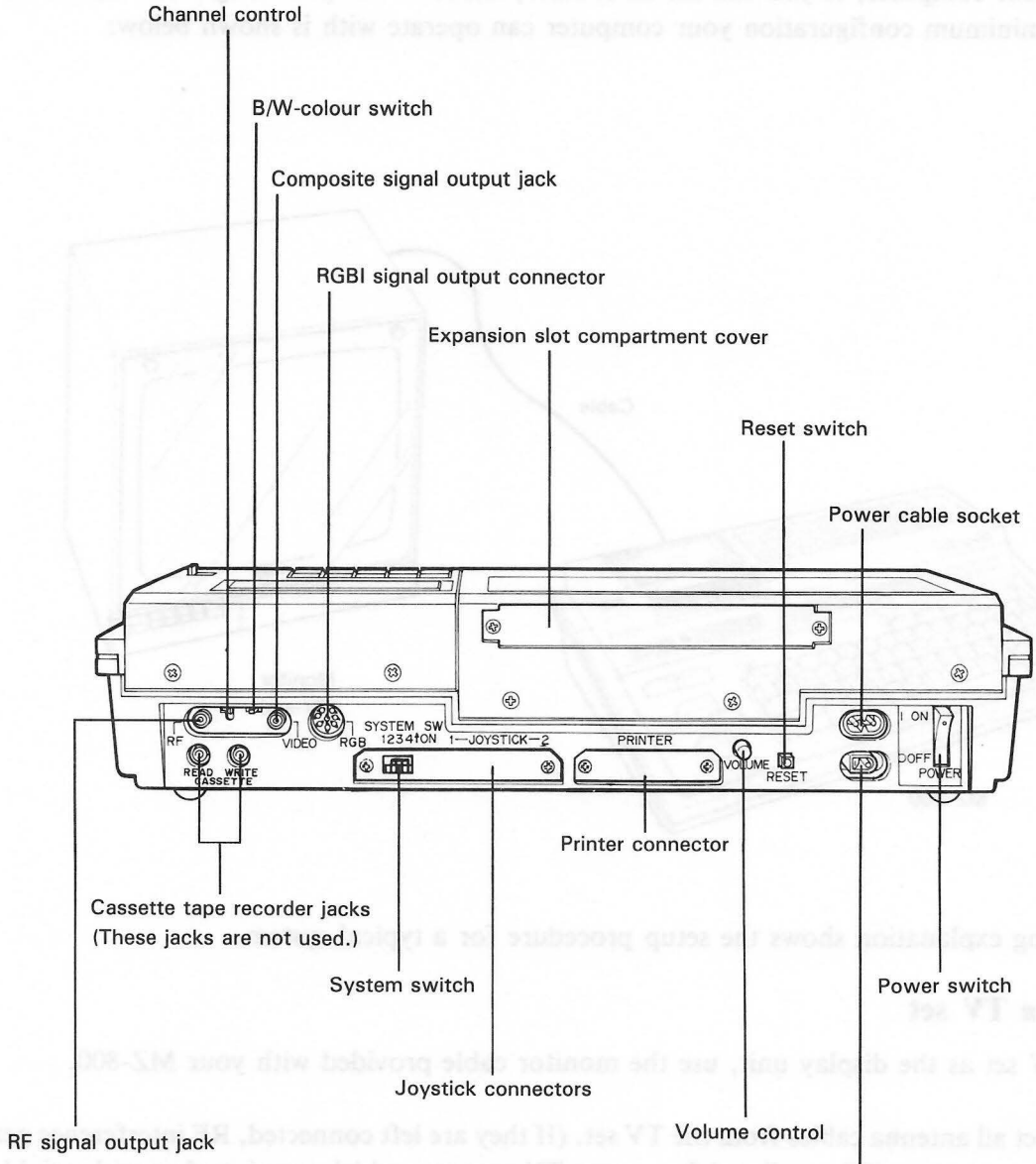
The explanations in this manual are based on the MZ-821.
However, the explanations on pages 7-3 and 7-18 apply only to the MZ-811.

1.3 Appearance

(Front view)

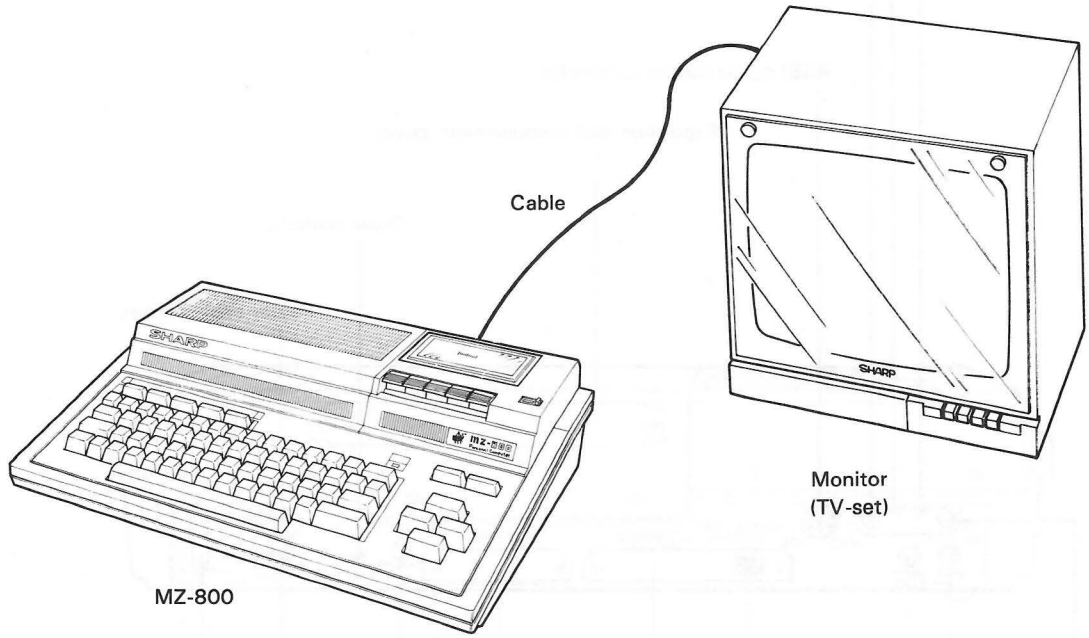


(Rear view)


Channel control
B/W-colour switch
Composite signal output jack
RGBI signal output connector
Expansion slot compartment cover
Reset switch
Power cable socket
RF
VIDEO
RGB
SYSTEM SW
1234↑ON
1--JOYSTICK--2
PRINTER
VOLUME
RESET
ON
OFF
POWER
READ
WRITE
CASSETTE
Printer connector
Cassette tape recorder jacks
(These jacks are not used.)
System switch
Power switch
Joystick connectors
RF signal output jack
Volume control
Printer power jack
(Note: If this jack is short-circuited, the memory contents will be lost.)

1.4 Setup

To operate your MZ-800 computer, you must first set up the system. To do this, you will need to connect a display unit to see what the computer is doing. SHARP supplies several types of display units for the MZ-800 computer, or you can use an ordinary home TV set providing it can receive the VHF band. The minimum configuration your computer can operate with is shown below:

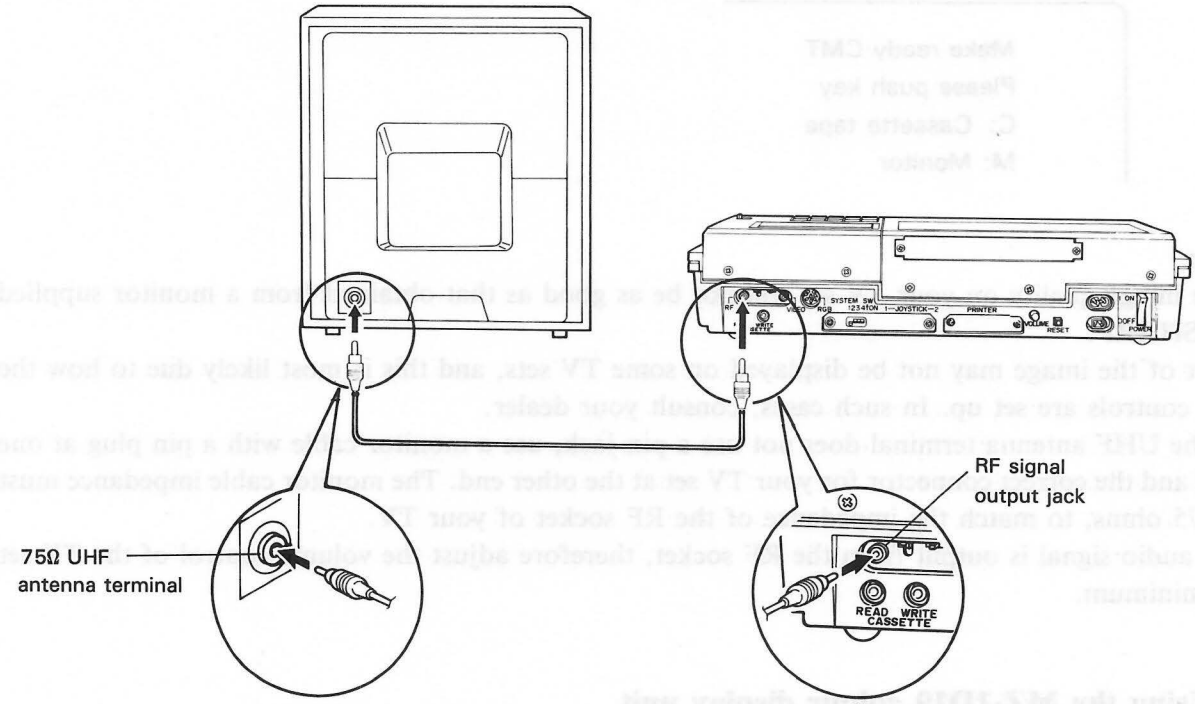


The following explanation shows the setup procedure for a typical system.

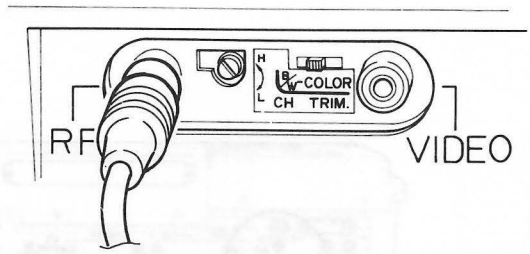
(1) Using a TV set

To use a TV set as the display unit, use the monitor cable provided with your MZ-800.

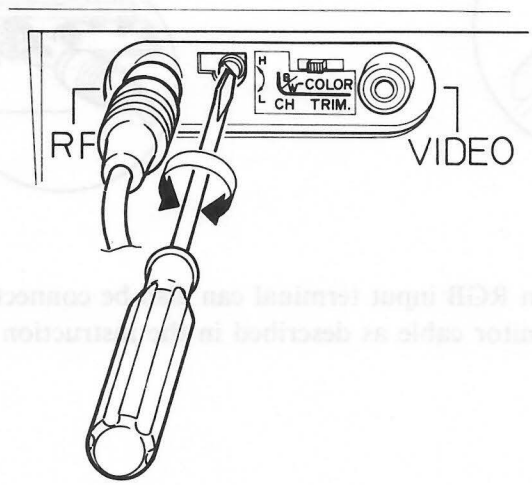
1) Disconnect all antenna cables from the TV set. (If they are left connected, RF interference generated by the computer will be radiated from your TV antenna, which may interfere with neighboring TV sets.)
2) Insert the monitor cable pin plug into the RF pin socket on the rear of the MZ-800. Connect the other end of the cable to the 75-ohm UHF antenna terminal on your TV set.



3) If the TV set is a colour unit, position the B/W-colour switch on the MZ-800 to COLOR, otherwise position the switch to B/W.



4) Tune the channel selector on your TV set to a vacant channel between 33 and 39.
5) Turn on the TV set then turn on the MZ-800. As shown in the figure below, adjust the channel control trimmer so that the following image is clearly displayed on the TV screen.



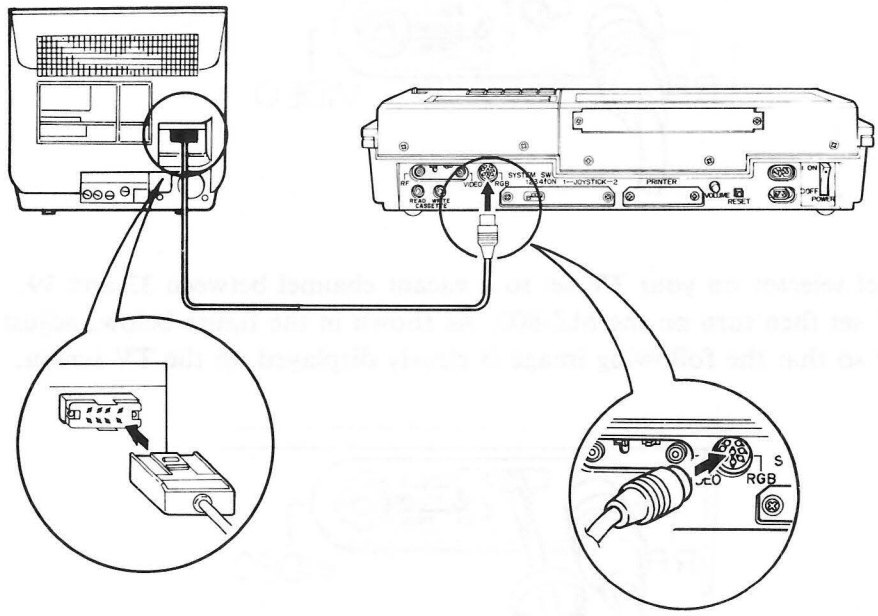
```
Make ready CMT
Please push key
C: Cassette tape
M: Monitor
```

Notes:

- The image quality on your TV set will not be as good as that obtained from a monitor supplied by SHARP.
- Part of the image may not be displayed on some TV sets, and this is most likely due to how the TV controls are set up. In such cases, consult your dealer.
- If the UHF antenna terminal does not use a pin jack, use a monitor cable with a pin plug at one end and the correct connector for your TV set at the other end. The monitor cable impedance must be 75 ohms, to match the impedance of the RF socket of your TV.
- No audio signal is output from the RF socket, therefore adjust the volume control of the TV set to minimum.

(2) Using the MZ-1D19 colour display unit

1) Plug the square connector of the connection cable provided with the MZ-1D19 into the connector on the rear panel of the display unit.
2) Plug the DIN connector of the connection cable into the RGB connector on the rear panel of the MZ-800.

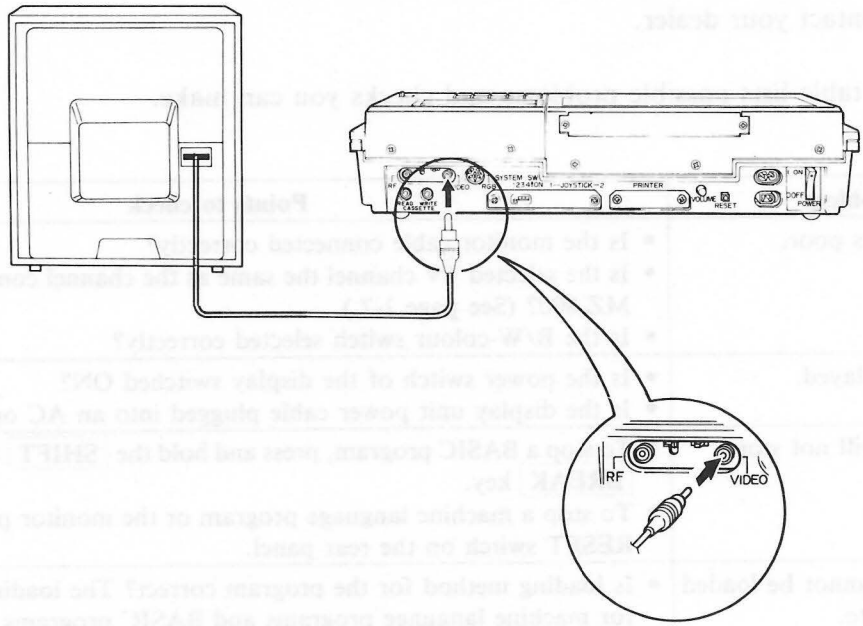


Note:

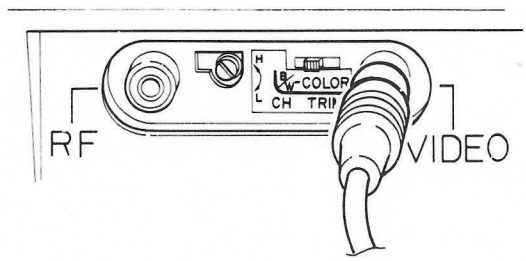
A colour TV set which has an RGB input terminal can also be connected to the RGB connector of the MZ-800. Prepare the monitor cable as described in the instruction manual for the TV set.

(3) Using a green display unit (MZ-1D04)

Insert the pin plug of the green display unit cable into the composite signal output jack on the rear panel of the MZ-800.



Position the B/W-colour switch to B/W.



Note:
A colour TV set with a video input terminal can be connected to the composite signal output jack of the MZ-800. The monitor cable provided with the MZ-800 can be used for this connection.

1.5 In Case of Difficulty

If you have any problems with your MZ-800 either now or in the future, read this section first then if necessary contact your dealer.

The following table lists possible problems and checks you can make.

Problem	Points to check
Image quality is poor.	• Is the monitor cable connected correctly? • Is the selected TV channel the same as the channel control setting on the MZ-800? (See page 1-7.) • Is the B/W-colour switch selected correctly?
Nothing is displayed.	• Is the power switch of the display switched ON? • Is the display unit power cable plugged into an AC outlet?
The program will not stop.	• To stop a BASIC program, press and hold the SHIFT key, then press the BREAK key. • To stop a machine language program or the monitor program, press the RESET switch on the rear panel.
The program cannot be loaded from the cassette.	• Is loading method for the program correct? The loading method differs for machine language programs and BASIC programs. Use the monitor L command to load a machine language program and the LOAD statement to load a BASIC program.
Other problems	• Press the RESET switch on the rear panel to restart MZ-800 operation.

Chapter 2 Start Up

2.1 Power-on

To start up your MZ-800 computer, first turn on the MZ-800, then turn on the display unit and any other connected peripheral devices power switch.

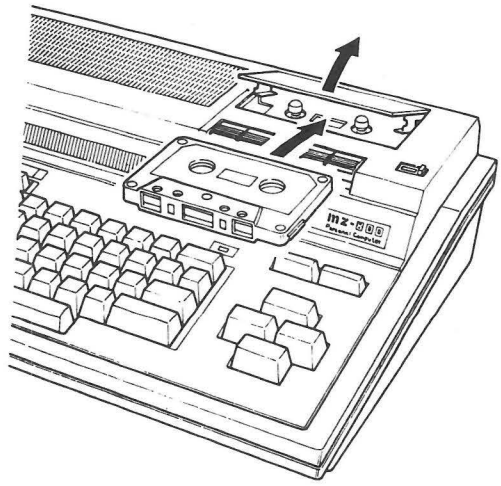
Turn on the equipment in the following order.

1) The MZ-800 computer
2) The expansion unit (MZ-1U06)
3) Peripheral devices (such as the printer)

You will see the following message on the screen of the display unit.

```
Make ready CMT
Please push key
C: Cassette tape
M: Monitor
```

Remove any slack from the cassette tape (see page 4-7). Press the [EJECT] button on the MZ-800 data recorder. Then insert the cassette with the side marked "BASIC 1Z-016" facing upwards.



Close the cassette compartment cover by hand. Press the [C] key on the main keyboard. (Pressing the [M] key starts the monitor. See Chapter 8.) The screen display will change as follows:

```
Make ready CMT
```

Press the [PLAY] button on the data recorder. The screen display will change as follows:

```
IPL is looking for a program
```

The following message is then displayed.

```
IPL is loading MZ-1Z016
```

Wait for several minutes, then the following display will appear on the screen. The tape stops automatically. Press the STOP button.

```
BASIC interpreter 1Z-016 VX.XX
Copyright (C) 1984 by SHARP CORP.

XXXXX bytes free
Ready
■
└─ Cursor (blinking)
```

This display indicates that the BASIC interpreter has been loaded into memory and the MZ-800 is ready to accept BASIC commands. This display is called the "initial" frame.

2.2 Power-off

When you switch the MZ-800 off, all programs and data stored in memory will be lost. Therefore, execute a SAVE operation prior to powering the computer off. (Chapter 3 describes how to save data onto the cassette tape.) To power off the MZ-800, finish any BASIC operations you may have started, then check the screen to make sure "Ready" is displayed and the cursor is blinking. Switch OFF the power switch.

Turn off the equipment in the following order.

1) Peripheral devices (such as the printer)
2) The expansion unit (MZ-1U06)
3) The MZ-800 computer

Note:
Do not power off the MZ-800 while the data recorder is operating (turning).

2.3 Running the Demonstration Program

The cassette provided with your MZ-800 contains a demonstration program, which can be executed by typing in the following after loading BASIC and advancing the tape until the counter reads 170.

RUN "CMT:" [CR]

When the screen display below appears:

RUN "CMT:"
↓ PLAY

Press the [PLAY] button.

The demonstration program will now be executed. To stop the program, press the [SHIFT] and [BREAK] keys at the same time. Press the [STOP] button after the tape has stopped.

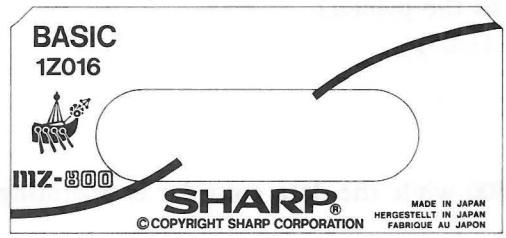
Note:
The tape will still move after the demonstration program has started.

- **Accessory Tape**

The accessory tape which is provided with the computer contains the following files.

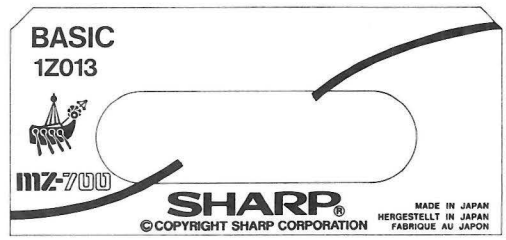
Side A
"MZ-1Z016" ..... MZ-800 BASIC Interpreter (1Z-016)
"OPENING 800" ..... Demonstration program for MZ-800 BASIC
"OPENING DATA" ..... Data for demonstration program

Side A label



Side B
"S-BASIC" ..... MZ-700 BASIC Interpreter (1Z-013)
"OPENING"
"MUSIC"
"COLOR PLOTTER" } ..... Demonstration programs for MZ-700 BASIC

Side B label



Chapter 3 Basic Operation

3.1 Introduction

Your MZ-800 has been encoded with a set of instructions that allow it to perform a variety of operations, such as accepting a command entered by you from the keyboard. This set of instructions is called the monitor program or simply the "monitor", and is stored in **ROM** (*). Any computer needs input from a human being to know what operation to perform next. After you power on the MZ-800 the monitor program makes the MZ-800 wait for you to input a command. Depending on the key you press, the monitor allows you to perform one of the monitor commands, or reads a larger set of instructions from an external memory device, such as the data recorder, and places it in **RAM** (*).

*: ROM and RAM are memory devices which store information for the computer. The ROMs (Read Only Memory) contain memory which can be read but cannot be changed or removed, even if the power is turned off. The RAMs (Random Access Memory) however, contain memory which can be both read and written. The MZ-800 uses ROM for storing the monitor program, and RAM for temporarily storing the BASIC interpreter, BASIC programs and data, and other information. The BASIC interpreter is explained in this chapter, while the monitor will be explained in detail in Chapter 8.

All the commands you input to your computer must be translated into the computer's own language, called **machine language**. Machine language consists of a collection of binary digits, which makes it extremely difficult for most people to understand. Luckily however, you need not worry about learning to understand machine language, since the **BASIC interpreter** does this for you. BASIC is a "high-level" language system which is similar to English and much easier for us as human beings to understand than machine language. The BASIC interpreter reads instructions written by you in BASIC and interprets them into the MZ-800's machine language.
If you press the [C] key when the initial frame is displayed, the monitor loads the BASIC interpreter into RAM from the cassette, the BASIC interpreter then begins operating. ("Load" means that information is read from one memory device, e.g., the cassette, and is placed in another memory device, e.g., the RAM.) Instructions written in BASIC are called **commands** or **statements**. The BASIC interpreter displays the following frame after the BASIC interpreter has been loaded.

```
BASIC interpreter 1Z-016 VX.XX
Copyright(C) 1984 by SHARP CORP.

 XXXXXbytes free
Ready
■
└— (blinking)
```

This display indicates that you can use the computer interactively, i.e., when you type in a command, the computer responds. If you type an incorrect command, the computer will answer with an error message.
Each command evokes only one response from the computer, and multiple commands are difficult to connect in a sequence. Because of this, you cannot get the computer to perform complicated operations in the interactive mode. The solution to enable the computer to perform complicated task, is to write a program and store it in RAM. A program is a series of statements which are automatically interpreted by the BASIC interpreter. A program which can be interpreted by the BASIC interpreter is called a BASIC program.

3.2 Getting to Know the Keyboard

Follow the start-up procedure described in Section 2.1, your MZ-800 is ready to accept commands typed in from the keyboard.

```
BASIC interpreter 1Z-016 VX.XX
Copyright(C) 1984 by SHARP CORP.

 XXXXXbytes free
Ready
■
(blinking)
```

The blinking block-shaped marker you can see on the screen is called the cursor. When you press any character key on the main keyboard, the cursor will move the right, with the typed character appearing in the previous cursor position.
Press other character keys, and the characters will appear in the order in which you type them. The cursor is always positioned to the immediate right of the character typed last. Next, press the key marked "CR" located on the right side of the main keyboard. You will see the message "Syntax error" appears on the next line.

```
BASIC interpreter 1Z-016 VX.XX
Copyright(C) 1984 by SHARP CORP.

 XXXXXbytes free
Ready
ASDFGJJ
Syntax error
Ready
■
```

The message "Syntax error" indicates that the computer cannot understand what you have just typed. This is because the computer only recognizes commands from the BASIC programming language. (Remember the BASIC interpreter?). BASIC will be explained more fully in Chapter 6, while "Syntax error" and other error messages are listed in Appendix L. At the moment, this and the following exercises don't require you to know anything about BASIC. Now, type the following sentence from the main keyboard.

```
PRINT "ABC"
```

After the closing quotation mark, press the [CR] key. The characters "ABC" will appear below the sentence you just typed. The computer displays these characters in reply to the BASIC command you entered. The command was the word "PRINT", which instructs the computer to redisplay the characters typed between the quotation marks.
Words in the BASIC language vocabulary which instruct the computer to perform an operation (such as PRINT) are called commands or statements.

3.3 Writing a Simple Program

Start up the BASIC interpreter following the procedures described in Section 2.1.

```
BASIC interpreter 1Z-016 VX.XX
Copyright(C) 1984 by SHARP CORP.

 XXXXXbytes free
Ready
■
(blinking)
```

Type the following characters.

```
10 CLS
```

Press the [CR] key. CLS is a statement which erases all the characters from the screen. However, the computer does not act on the statement immediately as in the interactive mode. This is because we are now writing a program, which causes the BASIC interpreter to store the statement in memory rather than execute it immediately. After you input the statement, the cursor blinks at the beginning of the line below "10 CLS" as shown below:

```
10 CLS
■
```

When a number precedes a statement, the BASIC interpreter stores the statement in memory. The number preceding the statement is called the line number, and when many statements are stored in memory, the line numbers indicate the order in which the statements are interpreted and performed by the computer. Type the following characters then press the [CR] key.

```
RUN
```

All characters will disappear from the screen. RUN is a command which orders the BASIC interpreter to interpret into machine language instructions all the statements stored in memory. The statements are interpreted in ascending order of the line numbers and given to the computer. The CLS statement is still in memory. You check this by typing the following:

```
LIST [CR]
```

[CR] indicates the [CR] key has to be pressed. The following display appears:

```
LIST
10 CLS
Ready
■
```

Type in the following.

```
20 PRINT "SAMPLE PROGRAM" CR
30 PRINT "MZ-800" CR
40 END CR
```

You already know that in this program the PRINT statement will redisplay the characters between quotation marks in lines 20 and 30. The END statement informs the BASIC interpreter of the end of program. Type in RUN and press the CR key, the screen will then reappear as shown below.

Note:
From now on, you will frequently see the phrase "Enter a command or statement" (e.g., "Enter the RUN command"). This actually means "Type in a command or statement, and press the CR key". You should remember this.

```
SAMPLE PROGRAM
MZ-800
Ready
■
```

Enter the LIST command to display the whole program.

```
SAMPLE PROGRAM
MZ-800
Ready
LIST
10 CLS
20 PRINT "SAMPLE PROGRAM"
30 PRINT "MZ-800"
40 END
Ready
■
```

Enter the following command.

```
NEW CR
```

Then clear the screen by entering the CLS statement.

```
CLS CR
```

Now enter the LIST command.

```
LIST CR
```

The program can no longer be listed, since the NEW command erased the program from memory. Entering the NEW command allows you to now write a new program in memory. Enter the following program.

```
10 INPUT "A=";A
20 INPUT "B=";B
30 C=A+B
40 PRINT "A+B=";C
50 END
```

This program will calculate the sum of the two values A and B input from the keyboard. Two new statements are used in this program. These are the INPUT statement and the LET statement (LET is represented by the equal "=" symbol on line 30).
The INPUT statement on line 10 reads whatever number (don't type characters) you type in from the keyboard and assigns it as the value of A. The INPUT statement on line 20 reads a second number typed in from the keyboard and assigns it as the value of B. The LET statement on line 30 calculates the sum of values A and B and assigns the result as the value of C. At the moment in our program, the letters A, B, and C each represent a numeric value. When letters of the alphabet are assigned values like this, they are called "variables". Enter the RUN command after typing in the above program. The screen will change as follows:

```
10 INPUT "A=";A
20 INPUT "B=";B
30 C=A+B
40 PRINT "A+B=";C
50 END
RUN [CR]
A=
```

Type in any number and press the [CR] key. The message "B=" will be displayed following the above.

```
A=35 [CR]
B=
```

Type in another number and press the [CR] key. The sum of A and B will now be displayed as follows.

```
A=35
B=23 [CR]
A+B=  58
```

The program shown here is very simple. If you like, you can write your own programs by combining some of the commands and statements which are explained in Chapter 6.

You may be confused by the words "statement" and "command". Both commands and statements control operation of the computer. The distinction between commands and statements is thatcommands are generally entered without line numbers and are executed immediately after they are entered. Statements however, are included in a program and are only executed when the program is started by the RUN command.

In practice, most commands and statements can be used both with or without line numbers, so the distinction between them is more traditional than qualitative.

3.4 Editing Programs

The BASIC interpreter makes it possible for you to edit a program which is in memory. Therefore, if you type in any incorrect character during programming, you can correct it easily.

You can edit any portion of a program when that part of the program is displayed on the screen. The program can be displayed by the LIST command. The cursor can be moved in any direction by the cursor control keys (marked with arrows) you can change or delete the character in the same location as the cursor, or insert characters before the character in the same location as the cursor.
The keys which allow you to edit programs are as follows.

[→] : Moves the cursor one character position right.
[←] : Moves the cursor one character position left.
[↑] : Moves the cursor one line up.
[↓] : Moves the cursor one line down.
[INST] : Moves the character on which the cursor is located and all characters following it on the same line to the right by one character position, and inserts a space at the cursor position. This makes it possible to insert any character at the cursor position. To insert more than one character, press the [INST] key the required number of times.
If the end of a program line reaches the right end of the display while inserting blanks with the [INST] key, you cannot insert any more blanks. In this case, press the [CR] key and execute the LIST command. The new program listing will include a row of blanks following the line, allowing you to insert more blanks.
[DEL] : Deletes the character at the location to the left of the cursor position and moves all characters following it on the same line to the left by one character position.
[SHIFT] + [INST] (CLR)
: Clears the screen. ("[SHIFT] + [INST]" is another way of saying "press and hold the [SHIFT] key, then press the [INST] key".)
[SHIFT] + [DEL] (HOME)
: Moves the cursor to the upper left corner of the screen.

Type in again the program shown in Section 3.3, but with the following intentional mistakes:

```
10 CLS
20 PRINT "SIMPLE PROGRAM"
30 PINTT "MZ-800"
40 END
```

To edit and correct the above program, the program must be listed on the screen. To do this, execute the LIST command.

(1) Replacing a letter
SIMPLE on line 20 should read SAMPLE. Move the cursor to the position of character I by using the cursor control keys, then press the [A] key. After changing I to A, press the [CR] key, and the cursor will be returned to the beginning of line 30.

(2) Inserting a letter
Move the cursor to I in PINTT and press the [INST] key. Press the [R] key to insert R between the characters P and I.

(3) Deleting a letter
Move the cursor to the second T in PRINTT and press the [DEL] key to delete it.
Press the [CR] key, and the cursor will move to the beginning of line 40.

(4) Adding a line

A new line can be added to any portion of the program. For instance, if we want to insert a line between line 10 and line 20, move the cursor to the beginning of the new line below line 40. Type "15 REM Editing sample [CR] ".

```
10 CLS
20 PRINT "SAMPLE PROGRAM"
30 PRINT "MZ-800"
40 END
15 REM Editing sample
```

You may have noticed that until we added line 15, all the line numbers have been in increments of 10. There is no real technical reason for doing this other than the fact that increments of 10 leave space for extra lines to be inserted if you want to change the program later on, and increments of 10 are easy to remember. With this in mind, line number 15 could have just as easily been any other number between 11 and 19 inclusive, but "15" is convenient since it still allows even further lines to be added if later program changes are made.

Enter the CLS command, then enter the LIST command to confirm that the new line is now inserted between lines 10 and 20.

```
LIST
10 CLS
15 REM Editing sample
20 PRINT "SAMPLE PROGRAM"
30 PRINT "MZ-800"
40 END
```

(5) Deleting lines

Any program line can be deleted by using the DELETE command. To delete lines 15 and 20, type:

```
DELETE 15-20 [CR]
```

Enter the LIST command. The program listing should appear as follows:

```
10 CLS
30 PRINT "MZ-800"
40 END
```

Typing a line number and pressing the [CR] key also deletes the line.

(6) Renumbering

Enter the RENUM command to return all the line numbers to increments of 10. (RENUM can also be specified to increment the line numbers by any other value.)

```
RENUM [CR]
LIST [CR]
10 CLS
20 PRINT "MZ-800"
30 END
```

(7) AUTO command

The AUTO command is a convenient feature which allows the computer to automatically generate/ line numbers for you, in increments of 10 or the specified value. For details, see Chapter 6.

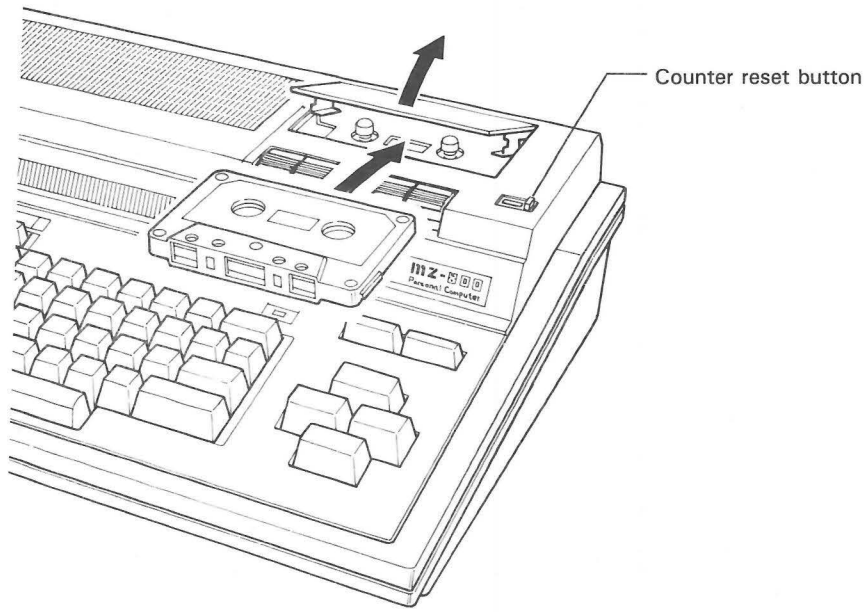
Remember to press the CR key after you finish editing each line; otherwise the editing changes for that line will not be entered into memory. Secondly, make sure that you move the cursor to the line below the last line of the program before typing RUN.

3.5 Saving a Program

When you turn the power switch off, any programs you may have typed will be lost. To reuse a program in a later session, you must save it onto any external storage device such as a cassette. The procedure for saving a program onto a cassette is described below.

- **Using a new cassette:**

1) Open the cassette compartment cover on the MZ-800 and insert the cassette. Close the cover, then press the counter reset button to reset the counter to "000".



2) Type the following.

SAVE "CMT:TEST" [CR]

This command instructs the computer to save the program in memory onto the cassette in the data recorder (the data recorder is indicated in the SAVE statement by CMT:). The program is saved with the name "TEST".

3) The next message to be displayed is "↓RECORD.PLAY". When you see this message, press the [RECORD] button.

4) When the message "Ready" appears on the screen and the tape stops, press the [STOP] button. Write down the program name "TEST" and the counter value at the end of the program on the cassette label.

- **Using a cassette which contains programs:**

When using a cassette which already contains programs, the counter value for the end of the program preceding the program you want to save must be known; otherwise your new program may become lost somewhere on the tape.

1) Insert the cassette into the data recorder and rewind the tape by pressing the [REWIND] button.

2) Press the counter reset button to reset the counter to "000".

3) Press the [FFWD] button. Stop the tape by pressing the [STOP] button when the counter value for the end of the preceding program nears.

4) Press the [PLAY] button to begin saving the program onto the cassette.

5) Perform steps 2 to 4 of the procedures described above for a new cassette.

6) When the program has been saved, note down the counter value, then run the tape an extra 2 or 3 counter revolutions and press the [STOP] button. When saving programs onto a cassette with existing programs, there is a possibility that the existing programs may be destroyed when the above procedures are performed. Therefore, it is recommended that you use a new cassette to save a program.

3.6 Loading a Program

The cassette provided with your computer contains a demonstration program, and you can also purchase commercially available programs. To use these programs, plus ones you may have written, you must load them off the cassette and into the computer's memory. The procedure for loading program is described below.

1) Insert the cassette which contains the program into the data recorder. Rewind the tape to the counter value of the program preceding the program you want if necessary.
2) Enter the following command to load the program into memory.

```
LOAD "CMT:<name of program>"
```

For example, to load the program "TEST",

```
LOAD "CMT:TEST"
```

3) When the message "↓PLAY" appears, press the [PLAY] button on the data recorder.
4) Press the [STOP] button when the tape stops.
5) To execute the program "TEST" now that it has been loaded, enter the RUN command when the message "Ready" is displayed on the screen.

```
Ready
RUN [CR]
```

Chapter 4 Keyboard and Data Recorder

4.1 Keyboard

4.1.1 Keyboard modes

The MZ-800 keyboard operates in one of the following modes:

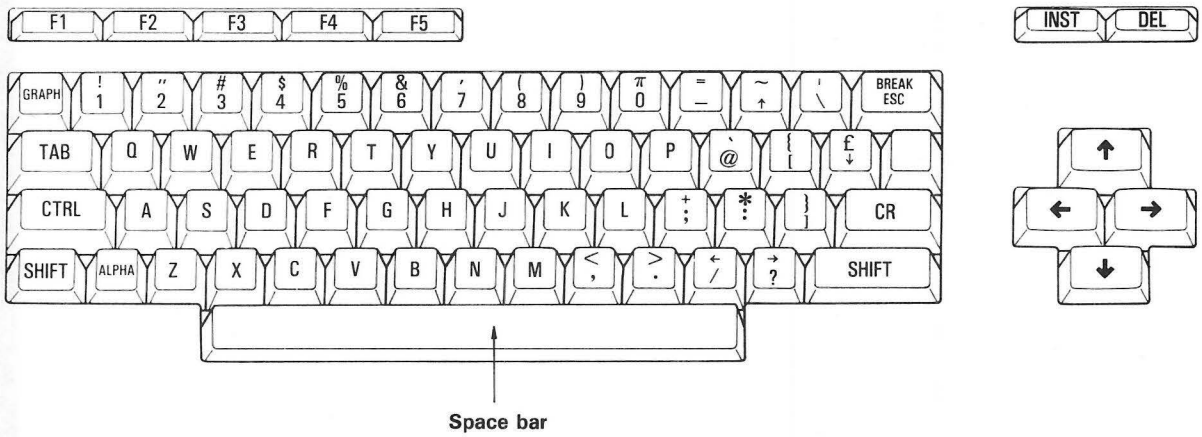
- Normal mode: Normally used to input the alphabetic characters, numbers and symbols. This mode is automatically set when the BASIC interpreter is started or the MZ-800 is reset.
- Shift lock mode: In this mode, all keys excepting [F1] through [F5] operate in the SHIFT mode. This mode is entered when [SHIFT] + [ALPHA] is pressed. Pressing [SHIFT] + [ALPHA] again resets the shift lock mode.
- Graphics mode: Used to input special graphic characters.

Three types of cursor are used to indicate the current keyboard mode.

■ : Normal mode cursor
● : Shift lock mode cursor
_ : Graphics mode cursor

4.1.2 Keys

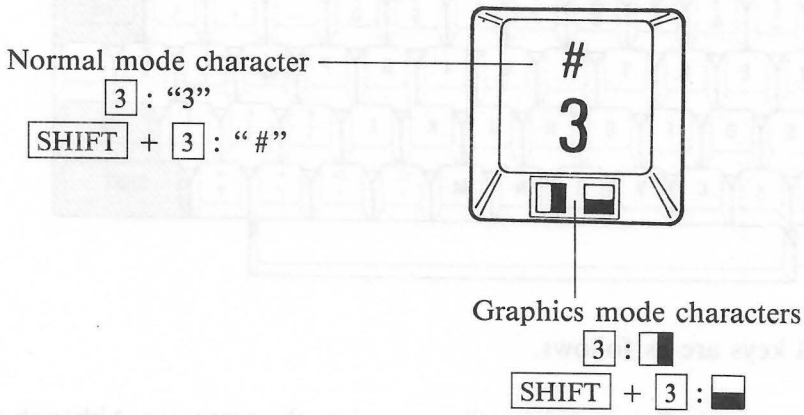
The keyboard has many keys and their functions are as follows:



(1) Character keys

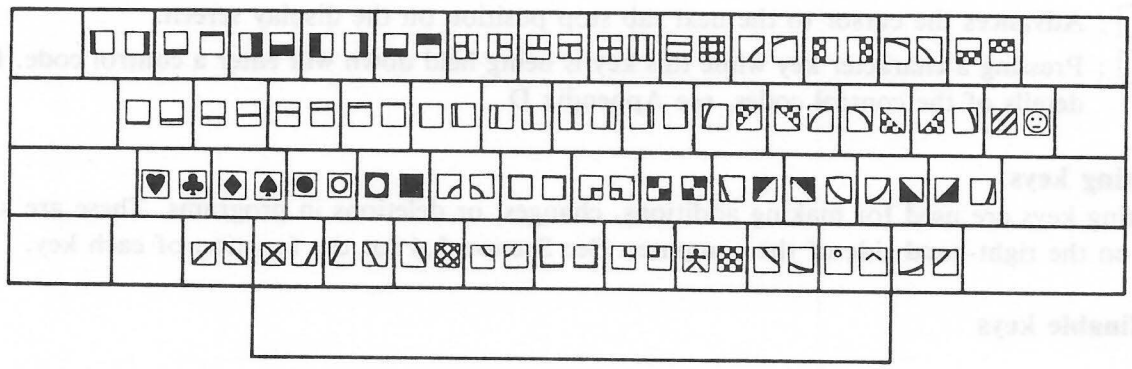
These keys are used to input letters, numerals and graphic characters.

Some character keys are marked with two different characters. These characters are input when the key is pressed in each of the input modes described above.

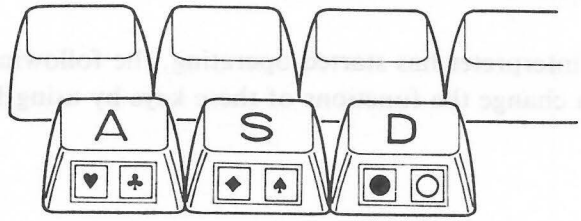


In the normal mode: when a character key is pressed, either the uppercase type for the letter marked on the keytop or the lower character on the keytop is input. When the character key is pressed together with the SHIFT key, either the lowercase type for the letter marked on the keytop or the upper character on the keytop is input.

In the graphics mode: each character key can be used to input either of two different graphic characters. When a character key is pressed by itself, the graphic character which is input is that shown on the left side of the corresponding keytop in the figure below. When it is pressed together with the SHIFT key, the graphic character which is input is that shown on the right side of the keytop in the figure. Pressing ↑, ←, →, ↓, CLR or HOME in this mode inputs ↑, ←, →, ↓, C or H.

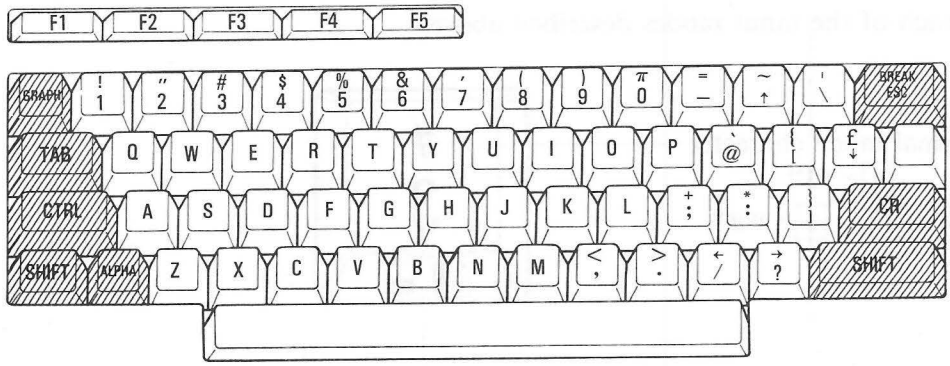


These graphic symbols are not printed on the keys. However, adhesive labels on which graphic symbols are printed are included with the MZ-800. You may find it convenient to stick the labels to the front of each key, as shown in the two figures.



(2) Special keys

These keys are used to control the computer and set the input mode for the character keys. The special keys are shaded in the figure below.



The functions of the special keys are as follows.

[CR] : This key is used to enter the line containing the cursor into the computer. Although characters typed by the character keys are displayed on the screen, the computer ignores them until the [CR] key is pressed.

[SHIFT] : This key operates keys in the shift mode while it is being held down.

[GRAPH] : Pressing this key switches the keyboard to the graphics mode.

[ALPHA] : Pressing this key returns the keyboard to the normal mode.

[BREAK ESC] : This key is used to input an ESC code.

[SHIFT] + [BREAK ESC]

: These keys are used to stop a program during execution or to stop cassette operation.

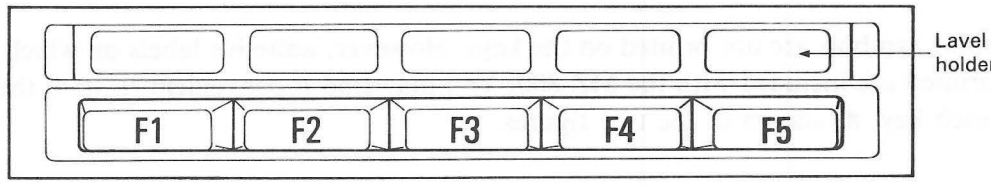
[TAB] : Advances the cursor to the next tab stop position on the display screen.

[CTRL] : Pressing a character key while this key is being held down will enter a control code. For details of the control codes, see Appendix D.

(3) Editing keys

The editing keys are used for making additions, changes, or deletions in programs. These are keys located on the right-hand side of the computer. See Section 3.4 for the function of each key.

(4) Definable keys



Immediately after the BASIC interpreter has started operating, the following functions are assigned to the definable keys. You can change the functions of these keys by using DEFKEY statement. See Chatper 6.

[F1] : "RUN␣␣␣" + CHR$ (13)
[F2] : "LIST␣"
[F3] : "AUTO␣"
[F4] : "RENUM␣"
[F5] : "COLOR␣"
[SHIFT] + [F1] : "CHR$("
[SHIFT] + [F2] : "DEF␣KEY("
[SHIFT] + [F3] : "CONT"
[SHIFT] + [F4] : "SAVE␣␣"
[SHIFT] + [F5] : "LOAD␣␣"

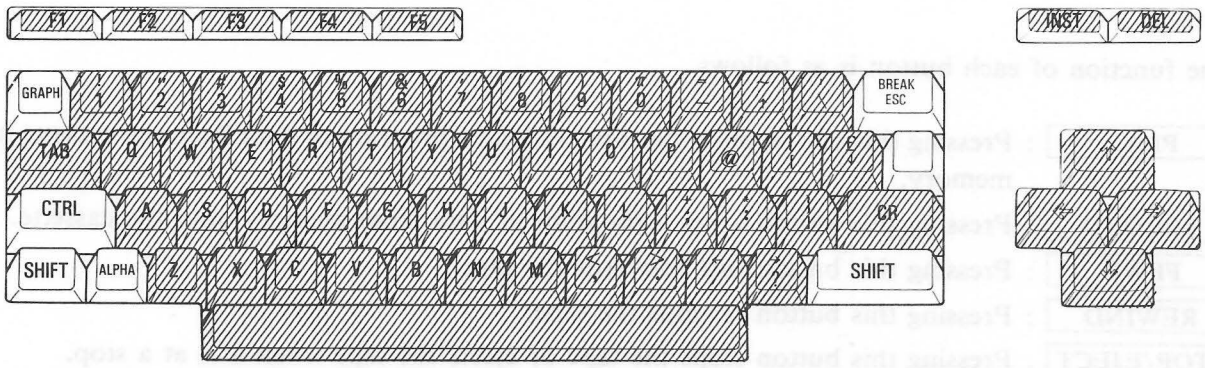
Note:
CHR$ (13) is the code for the [CR] key and "␣" represents a space.

- **Installing definable key labels**

You may find it convenient to insert the labels provided for the definable keys on which you write the assigned characters into the label holders located above the definable keys.
The labels can be inserted into this holder by pulling open the transparent label cover.

(5) Auto repeat function

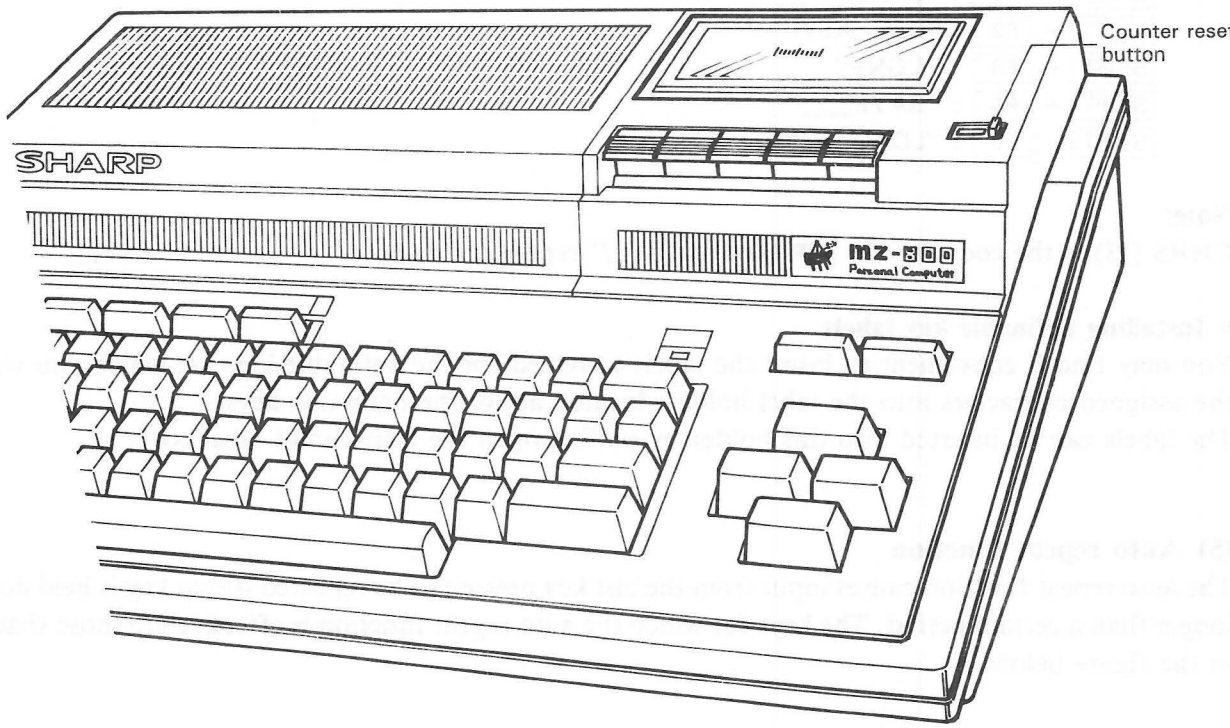
The auto repeat function causes input from the last key pressed to be repeated if that key is held down longer than a certain period. The keys for which the auto repeat function is effective are those shaded in the figure below.



4.2 Data Recorder

1) Hardware

The MZ-800 is equipped with a data recorder.



The function of each button is as follows.

PLAY	: Pressing this button plays the tape, to load a program or data from the cassette into memory.
RECORD	: Pressing this button saves a program or data from memory onto the cassette.
FFWD	: Pressing this button fast forwards the tape.
REWIND	: Pressing this button rewinds the tape.
STOP/EJECT	: Pressing this button stops the tape or ejects the tape when it is at a stop.

Counter reset button :

Pressing this button resets the counter to "000".

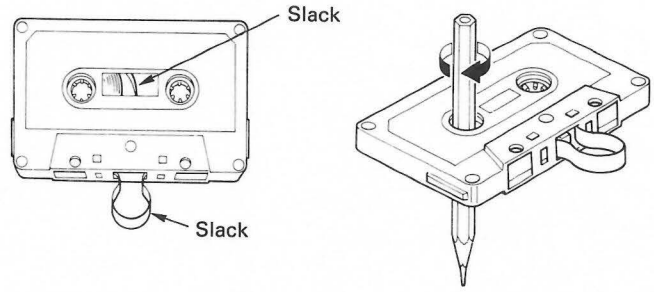
Note:

The FFWD and REWIND buttons are not automatically released when the tape end is reached. Be sure to press the STOP button when the tape end is reached.

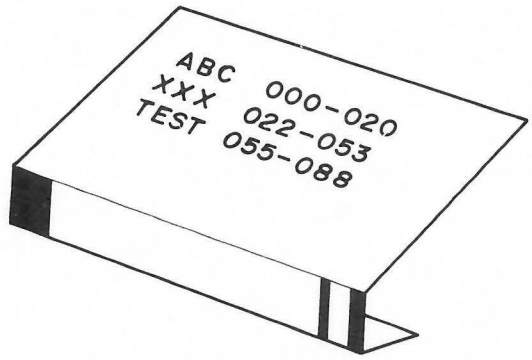
2) Tape handling

- Any commercially available cassette tape can be used with the MZ-800. However, it is recommended that you use quality cassette tape produced by a reliable manufacturer.
  * Use normal type tapes.
  * Avoid using C-120 type cassette tapes.
  * Use of C-60 or shorter cassette tapes is recommended.

- Be sure to remove slack from the tape by using a pencil or similar object before inserting the tape into the data recorder.



- Keep a record of the program name and the counter values for the beginning and end of each program after it has been saved.

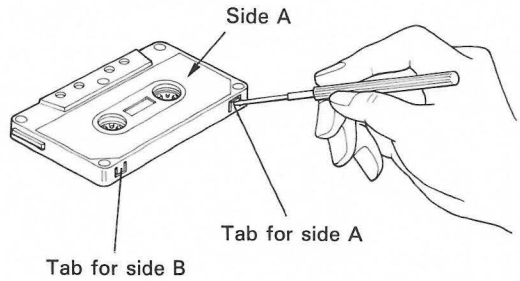


- Do not store cassette tapes near a TV set or speaker system which generates a magnetic field.

- **Protecting programs/data from accidental erasure**

To prevent data from being accidentally erased, remove the record lock-out tab from the cassette with a screwdriver or similar object. This will then make it impossible for the [RECORD] button to be pressed accidentally, thereby preventing erasure of valuable data or programs.

Remove the record lock-out tab with a screwdriver.



Chapter 5 Programming Concepts

This chapter describes fundamental concepts which will allow you to program the MZ-800 in BASIC. The information included in this chapter is essential to realizing the full potential of BASIC.

5.1 Multi-statement Lines and Line Numbers

As described in Chapter 3, a program consists of one or more program lines. Although each line of the examples in Chapter 3 contains only one statement, a program line can contain two or more statements, providing each statement is separated from each other by a colon (:). A program line which contains two or more statements is called a **multi-statement line**.

Example:

```
10 CLS:PRINT ''MULTI=STATEMENT'':END
```

Each program line begins with a **line number**. Line numbers can be any number between 1 and 65535. It is not necessary to specify line numbers consecutively, in fact, it is advisable to assign line numbers in increments of ten so that you can insert additional lines during program editing.

5.2 Numeric Data and String Data

Data handled by the computer is categorized into numeric data and string data. Broadly speaking, numeric data represents quantity or magnitude, whereas string data represents characters.

(1) Numeric data

The MZ-800 BASIC allows you to use numeric data in either decimal or hexadecimal notation. However, data in either notation is converted to binary form by the computer so that it can be stored in memory or used for calculations.

Decimal notation is probably the most familiar numbering system to you, and uses numerals from 0 to 9. Hexadecimal notation uses numerals from 0 to 9, then characters A to F to represent the values from 10 to 15. With this system, the number of significant digits required to express numbers increases by one each time the magnitude of the number being expressed increases by a factor of 16. Hexadecimal numbers are indicated by prefixing "$" to the character as follows.

$\$41 = 4 \times 16^1 + 1 \times 16^0 = 65$
$\$FA = 15 \times 16^1 + 10 \times 16^0 = 250$

Complements result for hexadecimal numbers greater than $7FFF.
For example, value resulting from $8000 is -32768 and that resulting from $FFFF is -1.

(2) String data

All characters are represented by numeric codes in the computer. These numeric codes are based on the ASCII code system. In this system, characters are represented by the numbers 0 to 255 or $00 to $FF. For example, the character "A" is represented by 65 (decimal) or $41 (hexadecimal).

5.3 Constants

(1) Numeric constants

Numeric constants are positive or negative numbers. They can be represented in either their ordinary form or an exponential form. Numeric constants must lie within the range 10^{-38} to 10^{38} (1E−38 to 1.7014118E+38), and the maximum number of significant digits is 8. If the value of a constant is out of the range, the result of operation is not assured.

Ordinary integers and decimal numbers are represented in their normal form as follows.

```
123
−123.4
+12
```

The "+" sign may be omitted for positive numbers.

Very large or small numbers are represented in the exponential form. In this form, a number is represented by a number representing the mantissa, E, and a number representing the exponent. Use of "E" in the exponential form is shown below.

```
1.23E+2
```

This represents $1.23 \times 10 \times 10 = 123$

```
−1.2E−1
```

This represents $-1.2 \div 10 = -0.12$

The "+" sign may be omitted for positive numbers. The mantissa must be less than 10 and greater than −10 and the exponent must be an integer between −38 and +38.

(2) String constants

A string constant is a set of characters enclosed in quotation marks (" "). The maximum number of characters in a string constant depends on the effective line length, but the total maximum number of characters of string data permitted is 255. Examples of string constants are as follows.

```
"ABC"
"12345"
"MZ-800"
```

Note:

Quotation marks are not required in DATA statements. (See Chapter 6.)

5.4 Variables

Variables are locations in memory which are used to hold values during program execution. You must give a specific name to each variable when writing a program. Values held by these variables may be arbitrarily changed during program execution.
There are three types of variables handled by MZ-800 BASIC: numeric variables, string variables and system variables.

(1) Numeric variables

Numeric variables can hold only numeric data. The name of each variable may be composed of any number of characters, but only the first two characters actually identify the variable. For example, AB and XYZ are different variables, but ABC and ABD are handled as the same variable.
Lowercase letters cannot be used for variable names.
The first character must be a letter from A to Z, but the second and the following characters may be any letter of the alphabet or numbers; however, special characters such as @ and * cannot be used.
No reserved words (see Appendix C.) may be used as the names of a variable. For example, PRINT and C@ cannot be used as the names of variables.
Each numeric variable contains 0 until some value is given.

(2) String variables

A string variable can hold only string data, and its name can be assigned in the same manner and with the same limitations as the name of a numeric variable. The only difference is that it is always followed by a dollar sign ($).
Each string variable may contain a maximum of 255 characters of string data. Each string variable includes only null characters until some stirng data is given.

(3) System variables

There are special variables called system variables, which are defined and used by the BASIC interpreter. The following table lists the system variables.

System variable	Explanation
TI$	Contains a 6-digit number which is the time from a 24-hour built-in clock. For example, the value "192035" indicates that the clock reads 19:20:35. The clock is always set to 00:00:00 when the power is turned on.
SIZE	Indicates the amount of free memory area which can be used for BASIC programs and data.
ERN	When an error occurs, this variable contains the corresponding error number.
ERL	When an error occurs, this variable contains the line number of the error.
CSRH	Contains the column position at which the cursor is located. $0 \leqq CSRH \leqq 39$ (40 column screen mode) $0 \leqq CSRH \leqq 79$ (80 column screen mode)
CSRV	Contains the line number at which the cursor is located. $0 \leqq CSRV \leqq 24$
POSH	Contains the X-coordinate of the graphics position pointer. $-16384 \leqq POSH \leqq 16383$
POSV	Contains the Y-coordinate of the graphics position pointer. $-16384 \leqq POSV \leqq 16383$

5.5 Array Variables

An array is an arrangement of variables of the same data type, which are referred to by a common name. Each variable of an array is identified by the common name, which is composed of a string formed in the same manner as a variable name and followed by subscripts enclosed within parenthese, e.g., A(X) and B$(x,y). An array with one subscript (such as A(X), B$(1) or P(100)) is called a one-dimensional array, while that with two subscripts (such as A(x,y), B$(1,3) or P(50,25)) is called a two-dimensional array. To use array variables in a program, the common name and the number of variables included in the array must be declared before they are used. For details see the explanation of the DIM statement in Chapter 6.

• Note Concerning Computational Error

Computational error must always be taken into consideration whenever a computer is used. The reason for this is that, although computational error can be reduced by increasing the number of digits of numerical data which are handled, not even a computer can handle an infinite number of digits. Further, the more digits are involved in any given calculation, the greater the amount of time which is involved in completing it.
Therefore, it is important to be aware of the sources of error and to construct programs so that error is minimized. (For example, use the sequence "5*6/3" instead of "5/3*6".)

Take the following into account when doing calculations in BASIC (1Z016) for the MZ-800.

(1) Rounding error
Rounding error is the error which results when the number of digits to the right of the decimal place exceed the number of effective digits which can be handled. For example, when the number 2/3 is calculated, the true result is 0.666666666 . . . (where the number of 6s is infinite). However, if the number of effective digits is 8, the result will be rounded to 0.66666667.

(2) Error resulting upon conversion to binary form
Although numbers are ordinarily input in decimal format, they are internally converted to binary form for calculation.
According, a binary number with an infinite number of digits may result upon conversion even if the original decimal number only has a few digits. For example, when the decimal number 0.1 is converted to binary form, the result is 0.00011001100 . . . . Since this must be rounded for calculation, a certain amount of error results.

(3) Increase in relative error due to subtraction
When one number is subtracted from another, the relative size of the error in the result will be greater than that in the original numbers. This is illustrated in the example below, where the digits which include error are marked with a dot (.). An error of ±1 in the number 100012 corresponds to an error percentage of about 0.001%; however, relative error is much greater after subtraction, since 11 ±1 corresponds to a relative error of about 10%.

$$\begin{array}{r} 100012\dot{} \\ -)\underline{100001\dot{}} \\ 11\dot{} \end{array}$$

(4) Error due to approximation
With a computer, exponentiation, trigonometric calculations, and logarithmic calculations are done using approximation; in consequence, a certain amount of approximation error results when such calculations are done.

5.6 Expressions

An expression is any combination of variables and constants which is combined with operators. Operators are symbols which perform mathematical or logical operations. The types of expressions handled by the MZ-800 BASIC are as follows.

- Arithmetic expressions
- String connective expressions
- Relational expressions
- Logical expressions

(1) Arithmetic expressions

An arithmetic expression consists of arithmetic operators, numeric constants, numeric variables and numeric functions. It calculates a numeric value from an operation(s) performed by the operator. (The numeric functions will be explained later in this chapter.)
The table below lists the arithmetic operators arranged in order of operational priority.

Arithmetic operator	Operation	Example
( )	Gives the highest priority to enclose operations.	(X + Y)
↑	Creates an exponentiatial value	X ↑ Y
−	Converts the sign of a value	−X
*, /	Multiplication, devision	X * Y, X / Y
+, −	Addition, subtraction	X + Y, X − Y

When an arithmetic expression includes operations of the same priority, they are performed in sequence from left to right.

(2) String connective expressions

String connective expressions are used to combine two or more data strings into a single string. A string connective expression consists of string constants, string variables, string functions and the operator "+". (The string function will be explained later in this chapter.)

Example:
"ABC" + "DEF" ..... "ABCDEF"
"A" + "B" + "C" ..... "ABC"

(3) Relational expressions

Relational expressions are used to compare two values and ascribe a logical value of either true (−1) or false (0) to the expression according to the result of the comparison. The result is used to make a decision regarding subsequent program flow. A relational expression can consist of constants, variables, arithmetic expressions, string connective expressions and relational operators.
The table below lists the relational operators.

Operator	Comparison	Example
=	Equal to	X=Y
<	Less than	X<Y
>	Greater than	X>Y
<=, =<	Less than or equal to	X<=Y, X=<Y
>=, =>	Greater than or equal to	X>=, X=>Y
<>, ><	Not equal to	X<>Y, X><Y

Note:
The relational values of character data are based on the characters' ASCII codes.

(4) Logical expressions

A logical expression expresses the Boolean sum or product of true or false values (−1 or 0) given by relational expressions. A logical expression is formed of logical values, relational expressions and logical operators. The following table lists the logical operators.

Operator	Meaning	Example
NOT	Logical negation	␣NOT X
AND	Logical product	X␣AND Y
OR	Logical sum	X␣OR Y
XOR	Exclusive OR	X␣XOR Y

Note:
Spaces indicated by ␣ must be included.

5.7 Files

A file is a program or a set of data which is output to or input from a peripheral device (such as a data recorder). A file is identified by a file descriptor, which consists of a name (called the file name) preceded by the name of the peripheral device (called the device name).

"<device name>:<file name>"

For example:
CMT:DEMO ..... The file named DEMO is output to or input from the data recorder.
RAM:TEST..... The file named TEST is output to or input from the RAM file board.

(1) File name
A file name can consist of up to 16 alphanumeric characters.

(2) Device name
The following table lists the device names which are used by the MZ-800 BASIC.

Device name	Device
CMT:	Data recorder
RAM:	Optional RAM file board
CRT:	Display device
LPT:	Printer
RS1:	RS-232C interface ports
RS2:	

5.8 Functions

(1) Numeric functions

Numerical functions such as SIN and COS perform arithmetic operations on given numeric expressions then return the result. The MZ-800 is provided with the following numerical functions.

ABS(X) — Absolute value
Returns the absolute value of numeric expression X.
Example: A = ABS(X). When X = 2.9, A = 2.9; when X = −5.5, A = 5.5.

SGN(X) — Sign
Returns 1, −1, or 0 according to whether numeric value X is greater than, less than, or equal to 0, respectively.
Example: A = SGN(X). When X = 0.4, A = 1; when X = −1.2, A = −1.

INT(X) — Integer
Returns the largest integer which is less than or equal to X.
Example: A = INT(X). When X = 3.87, A = 3; when X = 0.6, A = 0; when X = −3.87, A = −4.

SQR(X) — Square root
Returns the square root of X. The value specified for X must be greater than or equal to 0.
Example: A = SQR(X). When X = 4, A = 2.

EXP(X) — Exponential
Returns the value of the natural base e to the power of X.
Example: A = EXP(X)

Trigonometric Functions

SIN(X)
Returns the sine of X, where X is an angle in radians.
Use the following expression to obtain the sine of an angle in degrees.
SIN (X * π/180)
Example: A = SIN(X)

COS(X)
Returns the cosine of X, where X is an angle in radians.
Use the following expression to obtain the cosine of an angle in degrees.
COS (X * π/180)
Example: A = COS (X)

TAN(X)
Returns the tangent of X, where X is an angle in radians.
Use the following expression to obtain the tangent of an angle in degrees.
TAN (X * π/180)
Example: A = TAN(X)

ATN(X)
Returns the arc tangent of X in radians. The value returned is within the range $-\pi/2$ to $\pi/2$.
Use the following expression to obtain the arc tangent of X in degrees.
ATN (X) * 180/π
Example: A = ATN(X)

LOG(X) — Common logarithm
Returns the common logarithm of X ($\log_{10}X$), where X must be greater than 0.
Example: A = LOG(X)

LN(X) — Natural logarithm
Returns the natural logarithm of X ($\log_{e}X$), where X must be greater than 0.
Example: A = LN(X)

PAI(X) — Circular constant
Returns the value which is X times pi.
(PAI(1) = π = 3.1415927)
Example: A = PAI(X) or A = π * X

RAD(X) — Radian
Converts the numeric value X from degrees into a value in radians.
Example: A = RAD(X)

(2) Character functions

A character function processes character strings. The MZ-800 BASIC supports the following character functions. In the examples below, character variable A$ contains the character string "ABCDEFG".

LEFT$(x$,n)
x$: character string
n: numeric value (from 0 to 255)
Returns a string consisting of the left n characters of string X$.
Example: B$ = LEFT$(A$,2) produces string "AB"

MID$(x$,m,n)
x$: character string
m: numeric value from 1 to 255
n: numeric value from 0 to 255
Returns a string consisting of n characters following the mth character from the beginning of string x$.
Example: B$ = MID$(A$,3,3) produces string "CDE".

RIGHT$(x$,n)
x$: character string
n: numeric value (from 0 to 255)
Returns a string consisting of the right n characters of string x$.
Example: B$ = RIGHT$(A$,2) returns a string consisting of the right 2 characters of string A$. Therefore, variable B$ is returned as the string "FG".

Functions used with the PRINT statement

TAB(n)
n: numeric value
Moves the cursor to the (n + 1)th character position from the left end of the current line.
This function is ignored when n is less than the current cursor location.
Example: PRINT "A";TAB(3);"ABC"

```
A      A B C
0 1 2 3 4 5      ← column positions which are not displayed.
      ↑
      └──────── String 'ABC' is displayed from column 3.
```

SPC(n)

n: numeric value

Returns a string of successive spaces, the length of which is expressed by n.

Example: PRINT "A";SPC(3);"ABC"

```
A       A B C
0 1 2 3 4 5 6  ← column positions which are not displayed.
  3 spaces
```

(3) Numeric value/character string conversion functions

The following functions convert a numeric expression into a character string or vice versa.

STR$(n)

n: numeric value

Converts numeric value n into a character string.

(A hexadecimal value is preceded by $.)

Examples: A$ = STR$(−12)

The character string "−12" is returned as A$.

B$ = STR$(70 * 33)

The character string "2310" is returned as B$.

C$ = STR$(1200000 * 5000)

The character string "6E + 09" is returned as C$.

Note:

A positive integer displayed or printed is preceded by a single space which indicates that the plus sign (+) is valid but has been omitted. However, this space is deleted when the integer is converted into a string by the STR$ function.

VAL(x$)

x$: character string

Converts a character string into a numeric value.

Example: A = VAL ("123")

The string "123" is converted into the numeric value 123.

A = VAL ("$FF")

A string "$FF" is converted into the numeric value 255.

ASC(x$)

x$: character string

Returns the numeric value which is the ASCII code for the first character of string X$.

Examples: X = ASC("A")

Returns the numeric value 65, which is the ASCII code for character "A".

Y = ASC("SHARP")

Returns the numeric value 83, which is the ASCII code for the first character of the string "SHARP".

CHR$(n)

n: numeric value (greater than 32)

Returns the character whose ASCII code is integer expression n.

When a space is to be displayed, use PRINT "␣" or PRINT SPC(1).

Examples: A$ = CHR$(65)

Returns "A", which has an ASCII code of 65.

PRINT CHR$(107)

Displays the graphics character "⊠", which has an ASCII code of 107. Multiple ASCII codes can be specified as follows:

A$ = CHR$(65,66,67,68)

LEN(x$)

x$: character string

Returns the number of characters in string x$.

Example: A = LEN("ABC")

Returns the number 3, which is the number of characters in string "ABC".

(4) Random number functions

RND(n)

n: numeric value

This function returns a pseudo random number for a given numeric value.

* Pseudo random numbers are generated from values between 0.00000001 and 0.99999999.
* When the numeric value specified is greater than 0, the function gives the next pseudo-random number in the current sequence.
* When the numeric value is less than or equal to 0, RND generates a new pseudo-random number set whose initial value is determined by the value specified for X, and gives the first number of the new set. This makes an operation such as a simulation with random numbers repetitive.

Example:

To generate a random number which is an integer from N to M, use the following formula:

```
INT(RND(X)*(M-N+1)+N)
```

The following program draws a number of circles. The radius of the circles and the coodinates are given by the random number.

```
10 FOR A=1 TO 100
20 B=RND(1)*320
30 C=RND(1)*200
40 D=RND(1)*100
50 E=INT(RND(1)*4)
60 CIRCLE [E,0]B,C,D
70 NEXT A
80 END
```

(5) Joystick functions

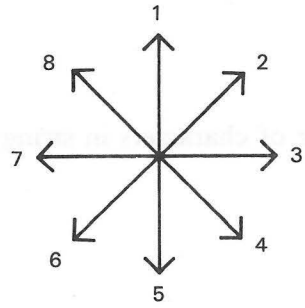
STICK(f)

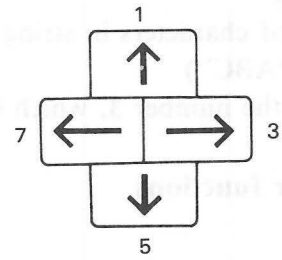
f: numeric value

Returns an integer from 1 to 8 which indicates the state of the joystick lever or the cursor control keys on the keyboard. The numeric value f specifies the device from which the data is read, as shown below.

0: Cursor contorl keys of the keyboard
1: Joystick 1
2: Joystick 2

The relationship between the integer and the direction in which the joystick lever is pushed (or the cursor control keys are pressed) is as follows:





Cursor control key

When the keyboard is selected by specifying 0 as f, integers 2, 4, 6, and 8 are returned when two cursor control keys are pressed at the same time, as shown below.

2: [↑] and [→]
4: [→] and [↓]
6: [↓] and [←]
8: [←] and [↑]

STRIG(f)

f: numeric value

Returns an integer 0 or 1 which indicates the state of the joystick button or the space bar on the keyboard. When the space bar on the keyboard or the joystick button is pressed, 1 is returned and when they are not pressed, 0 is returned. The integer value f specifies the device as follows:

0: Keyboard space bar
1: Joystick 1 button
2: Joystick 2 button

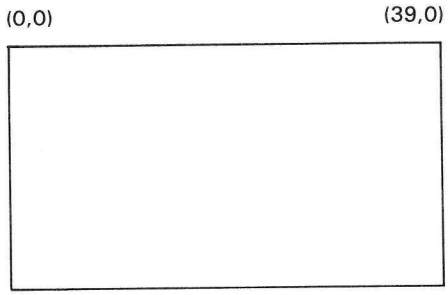
The following program uses STICK and STRIG functions. It draws a vertical, horizontal or inclinded line when a cursor key is pressed, and clears the screen when the space bar is pressed.

```
10 INIT ''CRT:M1''
20 A=STICK(0):B=STRIG(0)
30 ON A GOSUB 200,300,400,500,600,700,800,900
40 IF X<0 THEN X=0
50 IF X>319 THEN X=319
60 IF Y<0 THEN Y=0
70 IF Y>199 THEN Y=199
80 SET X,Y
90 IF B=1 GOTO 10
100 GOTO 20
200 Y=Y-1:RETURN
300 X=X+1:Y=Y-1:RETURN
400 X=X+1:RETURN
500 X=X+1:Y=Y+1:RETURN
600 Y=Y+1:RETURN
700 X=X-1:Y=Y+1:RETURN
800 X=X-1:RETURN
900 X=X-1:Y=Y-1:RETURN
```

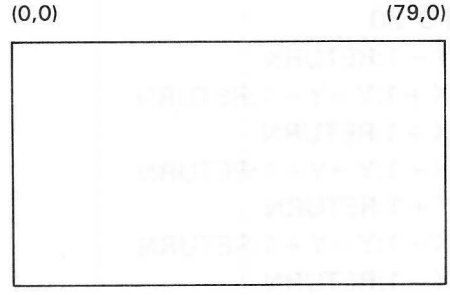
5.9 Screen Coordinates

Screen coordinates are used to specify the screen position in which characters and graphic data are to be displayed by display commands. Such coordinates are expressed in terms of a horizontal position and a vertical position. Character display positions are specified using character coordinates, and graphic display positions are specified using graphic coordinates.

- **Character coordinates**

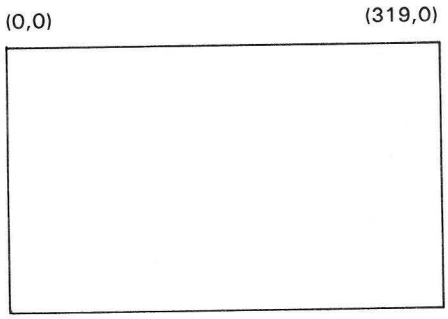


With 40 character line mode

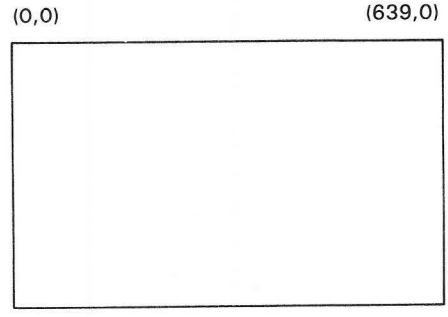


With 80 character line mode

- **Graphic coordinates**



320 × 200 mode



640 × 200 mode

The ranges of both character coordinates and graphic coordinates vary according to mode. The mode is specified with the INIT command.

Chapter 6 MZ-800 BASIC Commands and Statements

This chapter explains the MZ-800 BASIC (1Z-016) commands and statements. These commands and statements are functionally divided into the following eight groups.

- Fundamental commands
- Fundamental statements
- File control statements
- Graphics statements
- Music control statements
- Printer control statements
- Machine language control statements
- Error processing statements.

The commands and statements for the MZ-700 mode are summarized in Chapter 9.

Format Notations
The following rules apply to specification of commands, statements, and functions.

Angle brackets "< >" indicate items which must be specified by the user.

Items in square brackets "[ ]" are optional.

Items in { } are mutually exclusive; and only one of the items shown can be included when the statement is executed.

... indicates that the item preceding ... may be specified repeatedly.

6.1 Commands

AUTO

Format

AUTO [<starting line number>][,<increment>]

Abbreviated Format

A.

Explanation

The AUTO command automatically generates program line numbers during entry of BASIC program statements.
The default setting of both parameters is 10.

Example

(Example 1)
AUTO CR
10 ..... CR
20 ..... CR
30 ..... CR

(Example 2)
AUTO 300,5 CR
300 ..... CR
305 ..... CR
310 ..... CR

Example 2 automatically generates program line numbers, incrementing by 5 starting at line 300.

(Example 3)
AUTO 100 CR
100 ..... CR
110 ..... CR
120 ..... CR

Example 3 generates program line numbers with an increment of 10, starting at line 100.

(Example 4)
AUTO, 20 CR
10 ..... CR
30 ..... CR
50 ..... CR

Example 4 generates program line numbers with an increment of 20, starting at line 10.

Note:
The AUTO command is terminated by pressing SHIFT and BREAK .

DELETE

Format

DELETE [<starting line number> [−] <ending line number>]
DELETE <line number>

Abbreviated Format

D.

Explanation

Deletes program lines from <starting line Inumber> to <ending line number>.

Example

DELETE 150—350 [CR] ..... Deletes all program lines from 150 to 350.
DELETE —100 [CR] ..... Deletes all program lines up to line 100.
DELETE 400— [CR] ..... Deletes all program lines from 400 to the end of the program.
DELETE 150 [CR] ..... Deletes line 150.

LIST

Format

LIST [/P] [<starting line number>] [−] [<ending line number>]

Abbreviated Format

L.

Explanation

The LIST command lists on the display screen all or part of the program lines contained in the BASIC text area of the memory.
Output of the program listing to the display screen can be temporarily interrupted by pressing the space bar; listing is then resumed when the space bar is pressed again.
To terminate the listing, press the [SHIFT] + [BREAK] keys.
The program listing can be output to the printer by entering LIST/P.

Example

LIST [CR] ..... Lists the entire program.
LIST —30 [CR] ..... Lists all lines of the program up to line 30.
LIST 30— [CR] ..... Lists all lines of the program from line 30 to the end.
LIST 30—50 [CR] ..... Lists all lines of the program from line 30 to line 50.
LIST 30 [CR] ..... Lists line 30 of the program.

SEARCH

Format **SEARCH** [/P]<text data>

Abbreviated Format
SE.

Explanation The SEARCH command searches the BASIC program in memory for lines which contain the character string specified in <text data> and displays any found lines on the screen. When specifying a double quotation mark (") in <text data>, use CHR$(34).
Display of matching lines can be suspended by pressing the SPACE bar. Pressing the SPACE bar again will resume display. To terminate the SEARCH command, press SHIFT + BREAK . The /P option directs the output of the SEARCH command to the printer.

Example
SEARCH "ABC" ..... Searches for then displays on the screen the program lines that contain the character string "ABC".
SEARCH "PRINT"+CHR$(34)+"A"+CHR$(34) ..... Searches for program lines that contain PRINT "A".

RENUM

Format **RENUM** [<new line number>] [,<old line number>] [,<increment>]

Abbreviated Format
REN.

Explanation The RENUM command renumbers the lines of a BASIC program. When this command is executed, note that line numbers referenced in branch statements such as GOTO, GOSUB, ON~GOTO, and ON~GOSUB are also reassigned.

Example
RENUM ..... Renumbers the lines of the current program in memory so that they start with 10 and are incremented in units of 10.
RENUM 100 ..... Renumbers the lines of the current program in memory so that they start with 100 and are incremented in units of 10.
RENUM 100,50,20 ..... Renumbers lines of the current program in memory, which starts at line number 50. Line number 50 is renumbered to 100, and subsequent line numbers are incremented in units of 20.

(Before renumbering)
```
50 A=1
60 A=A+1
70 PRINT A
100 GOTO 60
```



(After renumbering)
```
100 A=1
120 A=A+1
140 PRINT A
160 GOTO 120
```

Note:
When specifying the new and old line numbers, the new line number specified must be larger than the old line number. Note that an error will result if execution of this command results in the generation of a line number which is greater than 65535.

NEW

Format

NEW

Explanation

The NEW command deletes programs in the BASIC memory area and clears program work areas such as the variables and arrays. When the BASIC area is limited with the LIMIT statement, the NEW command deletes only the programs in the BASIC area; it does not delete machine-language programs.

Example

```
10 INPUT A
20 PRINT A
30 END
```

When the above program is in memory, executing NEW will delete the program. (Confirm the deletion by using the LIST command.)

NEW ON

Format

NEW ON

Explanation

Expands the BASIC program area by deleting part of the BASIC interpreter which is relating to the plotter printer control. This command can be used only when the optional printer (MZ-1P16) is not used. This command deletes programs in the BASIC memory area.

Example

```
NEW ON..... Expands the BASIC program area.
```

CLR

Format

CLR

Explanation

The CLR command clears all variables and cancels all array definitions. All numeric variables are cleared to 0, all string variables are cleared to null strings (" ") and arrays are eliminated entirely by nullifying all previously executed DIM statements. Therefore, DIM statements must be reexecuted to redefine the dimensions of any required arrays before the arrays can be used again.

The CLR command also cancels all function definitions made with the DEF FN statement; therefore, it is necessary to reexecute DEF FN statements to redefine such functions before they can be used again.

The CLR command can not be included in a FOR~NEXT loop or subroutine.

Example

```
10 A=12
20 B$="parasol"
30 PRINT A,B$
40 CLR
50 PRINT A,B$
60 END
```

The CLR statement on line 40 clears variable A to zero and B$ to nulls.

CONT

Format **CONT**

Abbreviated Format

C.

Explanation The CONT command is used to resume execution of a program which has been interrupted by pressing [SHIFT] + [BREAK] or by a STOP statement in the program. When the message "Ready" is followed by a period (.), the CONT command can be used. Examples of situations in which the CONT command can and cannot be used are shown in the table below.

Program continuation possible	• Program execution stopped by pressing [SHIFT] + [BREAK] . • Program execution stopped by a STOP command.
Program continuation not possible	• Before a RUN command has been executed. • "Ready" is displayed due to an error occurring during program execution. • When cassette operation has been interrupted by pressing [SHIFT] + [BREAK] . • When program execution has stopped during execution of a MUSIC statement. • After program execution has stopped and "Ready" is displayed after execution of the END statement.

See also STOP

RUN

Format **RUN** [<starting line number>]

Abbreviated Format

R.

Explanation The RUN command executes the current program in the BASIC text area of memory. If the program is to be executed starting at the first program line, simply enter RUN and press the CR key. If execution is to begin with a line other than the lowest line number, type in RUN, <starting line number>, then press the CR key. When this command is executed with no <starting line number> specified, the BASIC interpreter clears all variables and arrays before passing control to the BASIC program.

Example

```
RUN ..... Executes the program from the beginning.
RUN 200..... Executes the program starting at line 200.
```

6.2 Fundamental Statements

CLS

Format **CLS**

Explanation The CLS statement clears the entire screen irrespective of the screen boundaries established by the CONSOLE command.

Example
```
10 CLS ..... Clears the entire screen.
```

See also CONSOLE

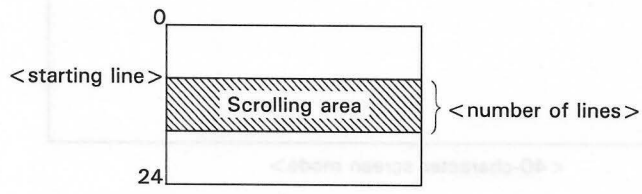
CONSOLE

Format **CONSOLE** [<starting line>,<number of lines>]

Abbreviated Format

CONS.

Explanation The CONSOLE command specifies the size of the scrolling area; i.e., the area which is cleared by specifying the CLS statement or pressing the SHIFT and CLR keys. This command becomes invalid after a PLOT ON command has been executed. Specify an appropriate value for the <number of lines> when editing; that is the <number of lines> must not be too small because it is harder to perform screen editing within a small scroll area.



Example
```
CONSOLE 0,25 or CONSOLE .. Scrolls the entire screen.
CONSOLE 5,15..... Scrolls the area between lines 5 and 15, inclusive.
```

CURSOR

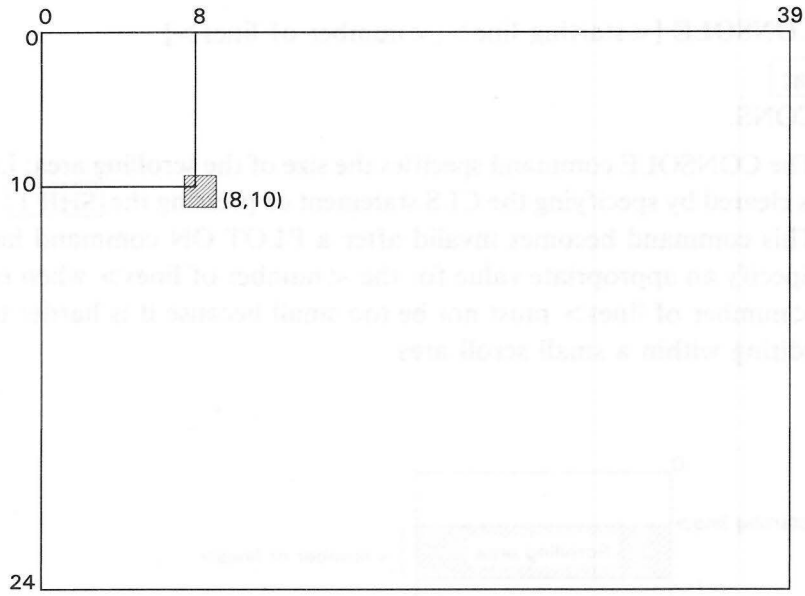
Format **CURSOR** <X-coordinate>, <Y-coordinate>

Abbreviated Format

CU.

Explanation The CURSOR statement moves the cursor to a specified position on the screen. It can be used together with the PRINT and INPUT statements to display characters at any desired location. The value of the <X-coordinate> must fall within the range for the screen mode specified in the INIT statement. The value of the <Y-coordinate> must be an integer from 0 to 24. If the value specified for either X or Y is other than an integer, it is converted to one by truncating the decimal fraction before the cursor is moved.

Example 10 CURSOR 8, 10 ..... Moves the cursor to point (8,10). After this statement is executed, when a PRINT or INPUT statement is executed the display will start at this point.



<40-character screen mode>

See also TAB, SPC

REM

Format

REM (remark)

Explanation

REM is a non-executable statement which is specified in a program line to cause the BASIC interpreter to ignore the remainder of that line. Since REM statements are non-executable, they may be included at any point in the program without affecting the program's execution. REM statements are generally used to make a program easier to read, or to add explanatory notes to a program.

Example

```
10 REM *** MZ-800 ***
```

LET

Format

LET <variable> = <expression>

Explanation

The LET statement assigns the value (numeric or string) specified by <expression> to the variable or array element specified by <variable>. As shown in the example below, LET may be omitted.

Example

```
10 A=10          10 LET A=10
20 PRINT A       20 PRINT A
30 END           30 END
```

The two programs above produce exactly the same result.

```
10 LET N=32
```

This statement assigns 32 to variable N.

```
10 LET A=A+5
```

This statement adds 5 to variable A.

```
10 LET B$="SUNDAY"
```

This statement assigns character string "SUNDAY" to character variable B$.

```
A=1 CR
```

This is an example of a command in the direct mode. 1 is assigned to variable A.

The following are examples of incorrect use of the LET statement.

```
20 LET A$=A+B.....
```

This is invalid because different types of variables (string and numeric) have been specified on either sides of the "=" sign.

```
20 LET LOG(LK)=LK+1 .....
```

Invalid because the left side of the statement is not a numeric variable or array element.

STOP

Format **STOP**

Abbreviated Format
S.

Explanation Temporarily stops program execution, displays the line number at which execution stops, then waits for the entry of executable commands in the direct mode.
The STOP statement is used to temporarily interrupt program execution, and may be inserted at as many points and locations in the program as required. Since execution of the program is only interrupted temporarily, the PRINT statement can be used in the direct mode to check the values stored in variables, after which execution can be resumed by entering CONT [CR] .

Example

```
10 READ A,B
20 X=A*B
30 STOP
40 Y=A/B
50 PRINT X,Y
60 DATA 15,5
70 END
RUN
Break in 30
Ready. ← This period indicates that the program can be continued by CONT.
```

Note:
Unlike the END statement, no files are closed by the STOP statement.

See also CONT

END

Format

END

Explanation

The END statement terminates program execution and returns the BASIC interpreter to the command mode for input of direct mode commands. When this statement is executed, "Ready" is displayed to indicate that the BASIC interpreter is ready. After the END statement has been executed, execution cannot be resumed by executing the CONT command even if there are executable statements on program lines following the END statement.

Note:
All open files are closed when the END statement is executed.

Differences between the STOP and END statements

	Screen display	Files	Resumption of execution
STOP	Break in ×××× Ready.	Open files are not closed.	Can be resumed by executing CONT.
END	Ready	Open files are closed.	Cannot be resumed.

FOR ~ NEXT

Format

FOR <control variable> = <initial value> **TO** <final value>
[**STEP** <increment>]
NEXT <control variable>

Abbreviated Format

F. ~ N.

Explanation

The FOR ~ NEXT statements repeat the instructions between the FOR and NEXT variables the specified number of times.

Example

```
10 A=0
20 FOR N=0 TO 10 STEP 2
30 A=A+1
40 PRINT ''N='';N,
50 PRINT ''A='';A
60 NEXT N
```

(1) In the program above, 0 is assigned to N as the initial value.

(2) Next, lines 20 through 50 are executed and the values of variables A and N displayed.

(3) In line 60, the value of N is increased by 2, after which the BASIC interpreter checks to see whether N is greater than 10, the final value. If not, lines following line 20 are repeated.

When the value of N exceeds 10, the program leaves the loop and the subsequent instructions (on lines following line 60) are executed. The program above repeats the loop 6 times.
If STEP <increment> is omitted from the statement specification, the value of <control variable> is increased by 1 each time the loop is repeated. In the program above, omitting STEP2 would result in 11 repetitions of the loop.

```
FOR  N=0  TO  10  STEP  2
       ↑      ↑       ↑
     Initial value  Final value  Increment
     of N           for N        for N
NEXT N
```

FOR ~ NEXT loops may be nested within other FOR ~ NEXT loops. When doing this, inner loops must be completely enclosed within outer ones, and not overlap. Also, separate control variables must be used for each loop.

Example

```
10 FOR X=1 TO 9
20 FOR Y=1 TO 9
30 PRINT X*Y;
40 NEXT Y
50 PRINT
60 NEXT X
70 END
```

Inner loop

Outer loop

```
10 FOR A=1 TO 3
20 FOR B=1 TO 5
30 FOR C=1 TO 7
    :
110 NEXT C
120 NEXT B
130 NEXT A
```

110 NEXT C,B,A

When loops C, B, and A all end at the same point as in the example above, one NEXT statement may be used to indicate the end of all the loops.

Incorrect example:

```
FOR J=1 TO 10
FOR J=K TO K+5
NEXT J
```

Different control variables must be used in each loop.

```
FOR I=1 TO 10
FOR J=K TO K+5
NEXT I
NEXT J
```

Loops may not overlap each other.

Note:

The syntax of BASIC does not limit the number of levels to which loops may be nested; however, space in the memory is required to store return addresses for each level, so the number of levels is limited by the amount of free memory space available. The CLR and LIMIT statements cannot be used within a FOR~NEXT loop.

LABEL

Format

LABEL "<label name>"

Abbreviated Format

LA.

Explanation

The LABEL statement defines a label. Labels are used to define the destination to which program execution will transfer from the GOTO or GOSUB statement. Proper use of labels in your program will substantially improve program readability.

Example

```
10 PRINT "SAMPLE"
20 GOSUB "ABC"
30 PRINT "END"
40 END
100 LABEL "ABC"
110 PRINT "LABEL SAMPLE"
120 RETURN
```

Line 100 defines the label "ABC" as the destination of the GOSUB statement on line 20. After the GOSUB statement on line 20 is executed, control is transferred to the subroutine starting at line 100.

See also

GOTO
GOSUB

GOTO

Format

GOTO { <line number> / <label> }

Abbreviated Format

G.

Explanation

The GOTO statement unconditionally transfers program execution to the line number specified in <line number> or <label>. If <line number> or <label> points to a line which contains executable statements (statements other than REM or DATA statements), execution resumes with that line; otherwise, execution resumes with the first executable statement following <line number> or <label>.

Example

```
10 N=1
20 PRINT N
30 N=N+1
40 GOTO 20..... Transfers program execution to line 20.
50 END
```

Since execution of the program shown above will continue indefinitely, stop it by pressing the SHIFT and BREAK keys together (this may be done at any time to stop execution of a BASIC program). To resume execution, execute the CONT command.

Note:
The line number specified in a GOTO statement may not be for a line inside a FOR~NEXT loop.

See also

LABEL
GOSUB

ON ~ GOTO

Format

ON <numerical expression> **GOTO** { <line number> [,<line number>] ... / <label> [,<label>] ... }

Abbreviated Format

ON ~ G.

Explanation

The ON ~ GOTO statement branches execution to one of the line numbers following GOTO, depending on the value of <numeric expression>. The value of <numeric expression> indicates which of the line numbers following GOTO will become the branch destination. Therefore, if <numeric expression> is 1, execution branches to the first line number in the list; if <numeric expression> is 2, execution branches to the second line number in the list, etc. For example:

```
100 ON A GOTO 200,300,400,500
```

Destination when
A is 1 = line 200
A is 2 = line 300
A is 3 = line 400
A is 4 = line 500

Example

```
10 INPUT''NUMBER'';A
20 ON A GOTO 50,60,70
30 GOTO 10
50 PRINT ''XXX'' : GOTO 10
60 PRINT ''YYY'' : GOTO 10
70 PRINT ''ZZZ'' : GOTO 10
RUN
NUMBER 1
XXX
NUMBER 2
YYY
NUMBER
```

If a decimal number such as 1.2 is specified, the decimal fraction is truncated before the statement is evaluated.

Note:
When the value of <numeric expression> in an ON ~ GOTO statement is greater than the number of line numbers specified following GOTO, execution continues with the next line of the program. This also applies if the value of <numeric expression> is less than 1 or negative.
Further, if the value of <numeric expression> is a non-integer, the decimal fraction is truncated to obtain an integer value before the statement is evaluated.

See also

GOTO
ON GOSUB

GOSUB ~ RETURN

Format

GOSUB { <line number> / <label> }
{
RETURN

Abbreviated Format

GOS. ~ RE.

Explanation

The GOSUB statement transfers program control to a subroutine identified with <label> or beginning at the line number specified in <line number>. After the subroutine has been executed, control is returned by the RETURN statement to the line following the GOSUB statement.

A subroutine is a set of statements that may be used more than once in a program. One subroutine may call another subroutine which may call still another subroutine. Nesting of such subroutines is limited only by the available memory space. Each called subroutine must have a RETURN statement at the end.

Example

```
10 INPUT A, B
20 GOSUB 100
30 B=C
40 GOSUB 100
50 PRINT C
60 END
100 C=A↑2+B
110 RETURN
```

10 — 60 Main program

100 — 110 Subroutine

ON ~ GOSUB

Format

ON <numeric expression> **GOSUB** { <line number>[,<line number>] ... / <label> [,<label>] ... }

Abbreviated Format

ON ~ GOS.

Explanation

The ON ~ GOSUB statement branches program execution to the subroutine indicated by one of the line numbers following GOSUB, depending on the value of <numeric expression>. The operation of this statement is basically the same as with the ON ~ GOTO statement, but all branches are made to subroutines. Upon return from the subroutine, execution resumes with the first executable statement following the ON ~ GOSUB statement which made the call.

Example

Let us try using the ON ~ GOSUB statement in a scheduling program. The most important point to note in the following program is that, a subroutine call is made at line 180, even though line 180 itself is part of a subroutine (from line 170 to 190) which in turn is called by line 90. Subroutines can be nested to many levels in this manner.

```
 10 A$=" ENGL  ": B$=" MATH  ": C$=" FREN "
 20 D$=" SCI   ": E$=" MUS   ": F$=" GYM "
 30 G$=" HIST  ": H$=" ART   ": I$=" GEOG "
 40 J$=" BUS   ": K$=" H RM  ": CLS
 50 INPUT "WHAT DAY?";X$
 60 FOR Z=1 TO 7:Y$=MID$("SUNMONTUEWEDTHUFRISAT", 1+3*(Z-1),3):
    IF Y$=X$ THEN X=Z
 70 NEXT Z
 80 FOR Y=0 TO 4: PRINT TAB(5+6*Y);Y+1;
 90 NEXT Y: PRINT
100 ON X GOSUB 180,120,130,140,150,160,170
110 PRINT: GOTO 50
120 PRINT "MON ";A$;B$;D$;G$;K$:RETURN
130 PRINT "TUE ";B$;E$;H$;H$;D$:RETURN
140 PRINT "WED ";C$;C$;I$;A$;F$:RETURN
150 PRINT "THU ";B$;D$;F$;G$;E$:RETURN
160 PRINT "FRI ";A$;D$;I$;C$;C$:RETURN
170 PRINT "SAT ";B$;G$;D$;K$:RETURN
180 FOR Y=1 TO 6
190 ON Y GOSUB 120,130,140,150,160,170
200 PRINT:NEXT Y
210 RETURN
```

IF ~ THEN ~ :ELSE

Format

IF { <relational expression> / <logical expression> } **THEN** { <statement> / <line number> / <label> }

[**:ELSE** { <statement> / <line number> / <label> }]

Abbreviated Format

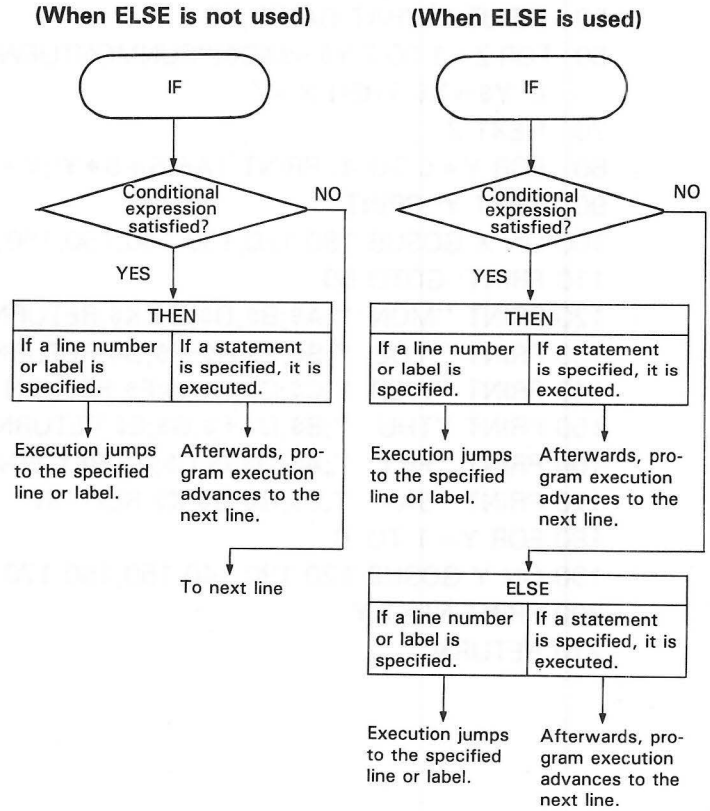
IF ~ TH. ~ :EL.

Explanation

IF ~ THEN ~ :ELSE statements are used to control branching of program execution according to the result of a logical or relational expression. When the result of such an expression is true, statements following THEN are executed. If a line number is specified following THEN, program execution jumps to that line of the program.

If the result of the logical or relational expression is false, statements following ELSE are executed. If a line number is specified following ELSE, program execution jumps to that line.

If :ELSE is omitted and the result of expression is false, execution continues with the next program line after the IF ~ THEN statement.



Example

```
10 IF C<1 THEN C=3 :ELSE C=C-1
```

This statement assigns 3 to C if C is less than 1; otherwise, assigns C-1 to C.

```
10 IF C< >D THEN 150 :ELSE END
```

This statement causes jump to line 150 if C is not equal to D; otherwise, ends program execution.

```
10 IF A$="ABC" THEN A$=A$+"DEF"
```

This statement assigns "ABCDEF" to A$ if A$ contains "ABC"; otherwise, the program proceeds to the next line.

Note:

(Precautions on comparison of numeric values with BASIC 1Z-016)

Numeric values are represented internally with binary floating point representation; since such values must be converted to other forms for processing or external display (such as in decimal format with the PRINT statement), a certain amount of conversion error can occur.

For example, when an arithmetic expression is evaluated whose mathematical result is an integer, an integer value may not be returned upon completion of the operation if values other than integers are handled while calculations are being made. Be aware of this and take it into consideration when evaluating relational expressions using "=".

This need is illustrated by the sample program below, which returns FALSE after testing for equality between 1 and 1/100*100.

```
10 A=1/100*100
20 IF A=1 THEN PRINT "TRUE":ELSE PRINT "FALSE"
30 PRINT "A=";A
40 END
RUN
FALSE
A= 1
```

The fact that both "FALSE" and "A=1" are displayed as the result of this program shows that external representation of numbers may differ from the number's internal representation in the computer.

IF ~ GOTO

Format

IF { <relational expression> / <logical expression> } **GOTO** { <line number> / <label> }

Abbreviated Format

IF ~ G.

Explanation

The IF ~ GOTO statement sequence evaluates the condition defined by <relational or logical expression>, then branches to the line number specified in <line number> or <label> if the condition is satisfied. As with the IF ~ THEN sequence, IF ~ GOTO is used for conditional branching. When the specified condition is satisfied, the program execution jumps to the line number specified in <line number> or <label>. If the condition is not satisfied, execution continues with the next line of the program. (Any statements following IF ~ GOTO on the same program line will be ignored.)

Example

```
10 T=0:N=0
20 INPUT "VALUE=";X
30 IF X=999 GOTO 100
40 T=T+X:N=N+1
50 GOTO 20
100 PRINT "**********"
110 PRINT "TOTAL:";T
120 PRINT "NO. ENTRIES:";N
130 PRINT "AVERAGE:";T/N
140 END
```

The above example gives the total and average of input values. If 999 is input, program execution is terminated.

See also

GOTO
IF ~ THEN ~ :ELSE
IF ~ GOSUB

IF ~ GOSUB

Format

IF { <relational expression> / <logical expression> } **GOSUB** { <line number> / <label> }

Abbreviated Format

IF ~ GOS.

Explanation

The IF ~ GOSUB statement evaluates the condition defined by <relational or logical expression>. If the condition is satisfied, the program execution branches to the subroutine beginning on the line number specified in <line number> or <label>. Upon completion of the subroutine, execution returns to the first executable statement following the calling IF ~ GOSUB statement. Therefore, if multiple statements are included on the line with the IF ~ GOSUB statement, execution returns to the first statement following IF ~ GOSUB.

Example

```
10 INPUT " X= ";X
20 IF X<0 GOSUB 100:PRINT "X<0"
30 IF X=0 GOSUB 200:PRINT "X=0"
40 IF X>0 GOSUB 300:PRINT "X>0"
50 PRINT "**********************"
60 GOTO 10
100 PRINT " * PROGRAM LINE 100 ":RETURN
200 PRINT " * PROGRAM LINE 200 ":RETURN
300 PRINT " * PROGRAM LINE 300 ":RETURN
```

See also

GOSUB ~ RETURN
IF ~ THEN ~ :ELSE
IF ~ GOTO

PRINT

Format

PRINT [<palette code>] <data> [{;,}<data>] ...

Abbreviated Format

?

Explanation

The PRINT statement displays data on the screen. <palette code> specifies the palette code for the colour of the text on the screen. If this code is omitted, the palette code specified in the colour statement is assumed.
When a semicolon is used to delimit two <data> items, it causes them to be displayed with no extra space. A comma, on the other hand, causes 10-character tabulations to be performed between the printout of each <data> item. If no <data> item is specified, this command performs a line feed.
Numeric data is displayed by this statement in one of two formats: real number format or exponential format. Numeric values in the range from 1×10^{-8} to 1×10^{8} are displayed in real number format; those beyond this range are displayed in exponential format.

Example

10 PRINT [2] "ABC";123 ... Displays the text data "ABC" and numeric data 123 with no space in the colour corresponding to palette code 2.

20 PRINT [3] "ABC",123 ... Displays the text data "ABC" and numeric data 123 with a 10-character tabulation between them. The colour assigned to palette code 3 is used.

Note:
Some special uses of the PRINT statement are shown below.
PRINT "C" Clears the entire screen and moves the cursor to the home position the upper left corner of the screen).
PRINT "H" Moves the cursor to the home position without clearing the screen.
PRINT "→" Moves the cursor one column to the right.
PRINT "←" Moves the cursor one column to the left.
PRINT "↑" Moves the cursor up one line.
PRINT "↓" Moves the cursor down one line.
To enter special characters for cursor control, press the GRAPH key; this changes the form of the cursor to "_". Next, press an edit key, CLR , HOME , → , ← , ↑ , or ↓ . After entering the special character, press the ALPHA key to return to the normal mode.

See also

COLOR
PAL

PRINT USING

Format **PRINT** [<palette code>] **USING** "<format string>"; <data> [{;,}<data>]...

Abbreviated Format

? USI.

Explanation

The PRINT USING statement displays data on the screen in a specific format. This statement should be entered using the same format as the PRINT statement, except for the specification of <format string>. <format string> consists of formatting characters which specify the format in which data is to be displayed, as described in the examples below.

(1) Formatting characters for numeric values

(a) #

A "sharp" symbol is used to represent each digit position. If the number to be displayed has fewer digits than positions specified, the number will be right-justified in the field.

```
10 A=123
20 PRINT USING "# # # # #";A
RUN
 123
```

(b) .

A period indicates the position in which the decimal point is to be displayed. The number of # signs to the right of the decimal point specifies the number of decimal places to be displayed.

```
10 A=12
20 PRINT USING "# # # .# #";A
RUN
 12.00
```

(c) ,

A comma placed at every third # sign in the <format string> parameter indicates the position in which a comma is to be displayed. Numbers will be displayed right-justified.

```
10 A=6345123
20 PRINT USING "#,# # #,# # #";A
RUN
6,345,123
```

(d) + and −

A plus (+) or minus (−) sign may be included at the end of <format string> to specify that the sign of the number is to be displayed in that position instead of a space. For instance, PRINT USING "# # # # +";A or PRINT USING "# # # # −";A will cause the sign to be displayed immediately after the number. (PRINT USING "# # # # −" causes a minus sign to be displayed following the number if the number is negative; if the number is positive, only a space is displayed in that position.) Furthermore, a plus sign may be specified at the beginning of a format string to indicate that the number's sign is to be displayed in the position regardless of whether it is positive or negative.

```
PRINT USING "# # # # +";−13
␣␣ 13−
PRINT USING "+ # # # #";25
␣␣ +25
```

Note:

Although a minus sign will be displayed if one is specified at the beginning of the format string, it will have no relationship to the sign of the number.

(e) * *

Specifying a pair of asterisks at the beginning of the format string indicates that asterisks are to be displayed in the positions of leading zeros.

```
10 A=123
20 PRINT USING "* * # # # #";A
RUN
* * *123
```

(f) ££

Specifying a pair of pound signs at the beginning of the format string indicates that a pound sign is to be displayed in the position immediately to the left of the number.

```
10 A=123
20 PRINT USING "££# # # #";A
RUN
␣␣ £123
```

(g) $$

Specifying a pair of dollar signs at the beginning of the format string indicates that a dollar sign is to be displayed immediately to the left of the number.

```
10 A=456
20 PRINT USING "$$# # # #";A
RUN
␣␣ $456
```

(h) ↑↑↑↑

Four exponential operators may be included at the end of a format string to control the display of numbers in exponential format.

```
10 A=51123
20 PRINT USING ''# #.# # #↑↑↑↑'';A
RUN
␣ 5.112E+04
```

In this case, the first number sign is reserved for display of the sign of the number.

(i) Extended list of operands

A list of variables may be specified following a single PRINT USING statement by separating them from each others with commas or semicolons. When this is done, the format specified in <format string> is used to display all resulting values.

```
10 A=5.3: B=6.9: C=7.123
20 PRINT USING ''# #.# # #''; {A;B;C}
                              {A,B,C}
RUN
␣ 5.300 ␣ 6.900 ␣ 7.123
```

The result is the same regardless of whether semicolons or commas are used to separate variables.

(2) Formatting characters for string values

(a) !

An exclamation mark in <format string> specifies that only the first character in the given string is to be displayed.

```
10 A$=''CDE''
20 PRINT USING ''!'';A$
RUN
C
```

(b) &␣␣␣␣&

Ampersands with n spaces between them specify that the first 2+n characters in the specified string are to be displayed. If the string is shorter than the field defined by <&␣␣␣␣&>, it will be left-justified in the field and padded with spaces on the right. If the string is longer than the field, the extra characters will be ignored.

```
10 A$=''ABCDEFGH''
20 PRINT USING ''&␣␣␣␣&'';A$
RUN
ABCDEF

10 A$=''XY''
20 PRINT USING ''&␣␣␣&'';A$
RUN
XY␣␣␣
```

(3) String constant output function

When any character other than those described above is included in the format string of a PRINT USING statement, that character is displayed together with the value specified following the semicolon.

```
10 A=123
20 PRINT USING ''DATA# # # #'';A
RUN
DATA ␣ 123
```

(4) Separating the USING clause

Usually, the keywords PRINT and USING are specified adjacent to eath other; however, it is possible to use them separately within the same statement.

```
10 A=−12 : B=14 : C=12
20 PRINT A;B; USING ''# # # #'';C
RUN
−12 ␣ 14 ␣␣ 12
```

In the above example, line 20 consists of a normal PRINT statement and a USING clause.

INPUT

Format

INPUT [<message>;]<variable>[,<variable>] ...

Abbreviated Format

I.

Explanation

The INPUT statement reads data entered during program execution and assigns it to <variable>.

When an INPUT statement is encountered during program execution, execution stops, a question mark appears, and the cursor blinks to indicate that the program is waiting for data. If <message> is specified, the message is displayed instead of the question mark. After data is typed in from the keyboard and CR is pressed, the data is assigned to <variable>, then program execution resumes. The types of the data and <variable> must be the same. Character constants can be entered without double quotes. In such cases, any leading or trailing spaces are ignored. However, if leading or trailing spaces or commas are to be included in the constant, enclose the entire character string in double quotes.

Example

```
10 INPUT A,B$ ..... Allows data to be entered and displays ?. When you
                     have entered the data, the program assigns the first
                     item to variable A and the second item to variable B$.
20 INPUT "A="; A..... Displays message "A=" and waits for data to be
                     typed in.
```

GET

Format

GET <variable>

Explanation

The GET statement checks whether any key on the keyboard is being pressed, and if so, assigns the key value to the variable specified in <variable>. The variable will be left empty (0 for a numeric variable or null for a string variable) if no keys are pressed.

With numeric variables, this statement allows a single digit (from 0 to 9) to be entered; with string variables, it allows a single character to be entered. Any non-numeric value entered for a numeric variable will be ignored.

Example

```
10 GET A$: IF A$="" THEN 10
20 PRINT A$
30 END
```

This program displays a character entered from the keyboard if the character is printable.

DIM

Format

DIM <variable>(<subscript>)[,<variable>(<subscript>)] ...
DIM <variable>(<subscript>,<subscript>)[,<variable>(<subscript>,<subscript>)] ...

Explanation

The DIM statement declares arrays with from one to four dimensions and reserves space in the memory for the number of dimensions declared (DIM: DIMENSION). Up to two alphanumeric characters beginning with an uppercase character can be specified for <variable> as the array name, and subscripts of any value may be specified to define the size of dimensions; however, the number of dimensions which can be used is limited in practice by the amount of free memory space available. Different names must be used for each array which is declared; for example, the declaration DIM A(5),A(6,3) is illegal. Execution of a DIM statement sets the values of all elements of the declared arrays to 0 (for numeric arrays) or nulls (for string arrays). Therefore, this statement should be executed before values are assigned to arrays.

If the DIM statement is executed on an array which has previously been declared, and if the newly declared dimensions are greater than the existing array, an error results.

All array declarations are nullified by execution of a CLR statement or a NEW statement.

Example

10 DIM A(3) ..... Declares 1-dimensional numeric array A with 4 elements.

A(0)	A(1)	A(2)	A(3)

3 + 1 = 4 elements

20 DIM B$(2,3)..... Declares 2-dimensional string array B with 12 elements.

B$(0,0)	B$(0,1)	B$(0,2)
B$(1,0)	B$(1,1)	B$(1,2)
B$(2,0)	B$(2,1)	B$(2,2)
B$(3,0)	B$(3,1)	B$(3,2)

(2 + 1) × (3 + 1) = 12 elements

```
10 DIM A(2)
20 FOR J=0 TO 2
30 INPUT A(J)
40 NEXT J
50 PRINT A(0), A(1), A(2)
60 END
```

Three array variables (A(0), A(1), and A(2)) are used in this example. The program inputs three numbers into these variables, then displays these numbers.

READ ~ DATA

Format

READ <variable>[,<variable>] ...
}
DATA <constant>[,<constant>] ...

Abbreviated Format

REA. ~ DA.

Explanation

Like the INPUT and GET statements, the READ statement is used to submit data to the computer for processing. However, unlike the other two statements, data is not entered from the keyboard, but is instead held in the program itself with DATA statements. More specifically, the function of the READ statement is to read successive items of data into variables from a list of values which follows a DATA statement. When doing this, there must be a one-to-one correspondence between the variables of the READ statements and the data items specified in the DATA statements. Quotation marks can be omitted for string data in DATA statements. However, they cannot be omitted for null strings and strings including spaces.

Example

(Example 1)

```
10 READ A,B,C,D
20 PRINT A;B;C;D
30 END
40 DATA 10,100,50,60
RUN
 10 100 50 60
```

In this example, the values specified in the DATA statement are read into variables A, B, C, and D by the READ statement, then the values of those variables are displayed.

(Example 2)

```
10 READ X$,A1,Z$
20 PRINT X$;A1;Z$
30 END
40 DATA A,1,C
```

As shown by the example above, string data included in DATA statements does not need to be enclosed in quotation marks.

```
RUN
A 1C
```

The READ statement in this example picks successive data items from the list specified in the DATA statement, then substitutes each item into the corresponding variable in the list following the READ statement.

(Example 3)

```
10 DIM A(2)
20 READ A(0),A(1),A(2)
30 PRINT A(0);A(1);A(2)
40 END
50 DATA 3,4,5
RUN
 3 4 5
```

The READ statement in this program substitutes the numeric values following the DATA statement into array elements A(0), A(1), and A(2), then the PRINT statement in line 30 displays the values of those array elements.

(Example 4)

```
10 READ A
20 READ B
30 DATA X
```

The example above is incorrect because firstly a numeric variable is specified by the READ statement on line 10, but the value specified following the DATA statement is a string value, and secondly there is no data which can be read by the READ statement on line 20.

See also RESTORE

RESTORE

Format

RESTORE [{<line number> / <label>}]

Abbreviated Format

RES.

Explanation

When the RESTORE statement is executed with no line number or only a line number of 0 specified, it causes the BASIC interpreter (when READ statements are encountered) to read the lists of data items from the beginning of the DATA statement with the smallest line number. If either <line number> or <label> is specified, this statement causes the BASIC interpreter to start reading data items in the DATA statement specified by the <line number> or <label> parameter or the subsequent DATA statement having the smallest line number.

Example

```
10 DATA ''PERSONAL COMPUTER''
20 DATA ''MZ-800''
30 READ A$,B$
40 PRINT A$;B$
50 RESTORE 20
60 READ C$
70 PRINT C$
80 RESTORE
90 READ D$
100 PRINT D$
110 END
RUN
PERSONAL COMPUTER MZ-800
MZ-800
PERSONAL COMPUTER
```

See also

READ ~ DATA

DEF FN

Format

DEF FN <function name>(<variable>)=<numeric expression>

Explanation

The DEF FN statement is used to define user function. Such functions consist of combinations of functions which are intrinsic to BASIC. The <function name> is an uppercase letter.

Example

DEF FNA(X)=2*X↑2+3*X+1 ..... Defines $2X^2 + 3X + 1$ as FNA(X).
DEF FNE(V)=1/2*M*V↑2..... Defines $1/2MV^2$ as FNE(V).

(incorrect definitions)

10 DEF FNK(X)=SIN(X/3+π/4), FNL(X)=EXP(−X↑2/K)
..... Only one user function can be defined by a single DEF FN statement.

10 DEF FND(X)=FNB(X)/C+X..... Any functions which have been defined with DEF FN cannot be used in another DEF FN.

Find the kinetic energy of a mass of 5.5 kg when it is imparted with initial accelerations of 3.5 m/s^2, 3.5×2 m/s^2, and 3.5×3 m/s^2.

```
10 DEF FNE(V)=1/2*M*V↑2
20 M=5.5:V=3.5
30 PRINT FNE(V), FNE(V*2), FNE(V*3)
40 END
```

Note:
All user function definitions are cleared when the CLR or NEW statement is executed.

TRON

Format **TRON[/P]**

Abbreviated Format

TR.

Explanation The TRON command traces the execution of the program. Once a TRON command is executed, line numbers of program lines are printed on the screen, enclosed in brackets ([ ]), as they are executed by the BASIC interpreter. The /P option directs the output of the TRON command to the printer.

Example

```
10 DEF FNA(X,Y)=X*Y
20 READ A1,A2,A3,A4
30 W=FNA(A1,A2):GOSUB 100
40 W=FNA(A2,A3):GOSUB 100
50 W=FNA(A3,A4):GOSUB 100
60 DATA 4,5,6,7
70 END
100 IF W>20 THEN PRINT"ABCD"
110 RETURN
```

Enter TRON before running this program.

```
RUN
[10][20][30][100][110][40][100]ABCD
[110][50][100]ABCD
[110][60][70]
```

Line numbers of program lines are printed as they are executed so you can keep track of how program execution proceeds. To terminate tracing, enter the TROFF command.

See also TROFF

TROFF

Format **TROFF**

Abbreviated Format

TROF.

Explanation The TROFF command disables the trace function.

See also TRON

DEF KEY

Format

DEF KEY(<key number>**)="**<character string>**"**

Explanation

Character strings can be assigned to any of the ten function keys to allow the strings to be entered at any time, simply by pressing a single definable function key.
Function key numbers 1 to 5 are entered just by pressing the corresponding function key at the top left corner of the keyboard, while keys 6 to 10 are entered by pressing the SHIFT key together with the corresponding function key. The function key number (1 to 10) is specified in <key number>, and the string or command which is to be assigned to the key is specified in <character string> exactly as you want it to appear. <character string> can be up to 15 characters long including spaces.
Execution of the DEF KEY statement cancels any existing function key definition.

Example

```
10 DEF KEY(1)="SHARP" ..... Defines key [F1] as SHARP.
20 DEF KEY(2)="RUN"+CHR$(13) ..... Defines key [F2] as RUN [CR].
```

Note:
CHR$(13) is the ASCII code for CR, which can be specified together with the string assigned to a definable function key to the same effect as you actually press the [CR] key.

KEY LIST

Format

KEY LIST

Abbreviated Format

K.L.

Explanation

The KEY LIST command displays a list of the character strings assigned to the definable function keys.

Example

```
KEY LIST [CR]
DEF KEY( 1)="RUN ␣␣␣"+CHR$(13)
DEF KEY( 2)="LIST ␣"
DEF KEY( 3)="AUTO ␣"
DEF KEY( 4)="RENUM ␣"
DEF KEY( 5)="COLOR ␣"
DEF KEY( 6)="CHR$("
DEF KEY( 7)="DEF ␣ KEY("
DEF KEY( 8)="CONT"
DEF KEY( 9)="SAVE ␣␣"
DEF KEY(10)="LOAD ␣␣"
Ready
```

The list above shows the initial settings for the definable keys.

Note:
"␣" indicates a space.

INIT

Format

(1) **INIT** "RAM:[<number of bytes>]"

(2) **INIT** "LPT: [M{0 / 1 / 2}] [, Sn] [, CR code]"

(3) **INIT** "RS{1 / 2}: <monitoring code>,<initialization code>[,<end code>]"

(4) **INIT** "CRT:[M<mode>][,B<block code>]"

Explanation

The INIT command defines the initial settings and modes for external devices.

(Format 1)

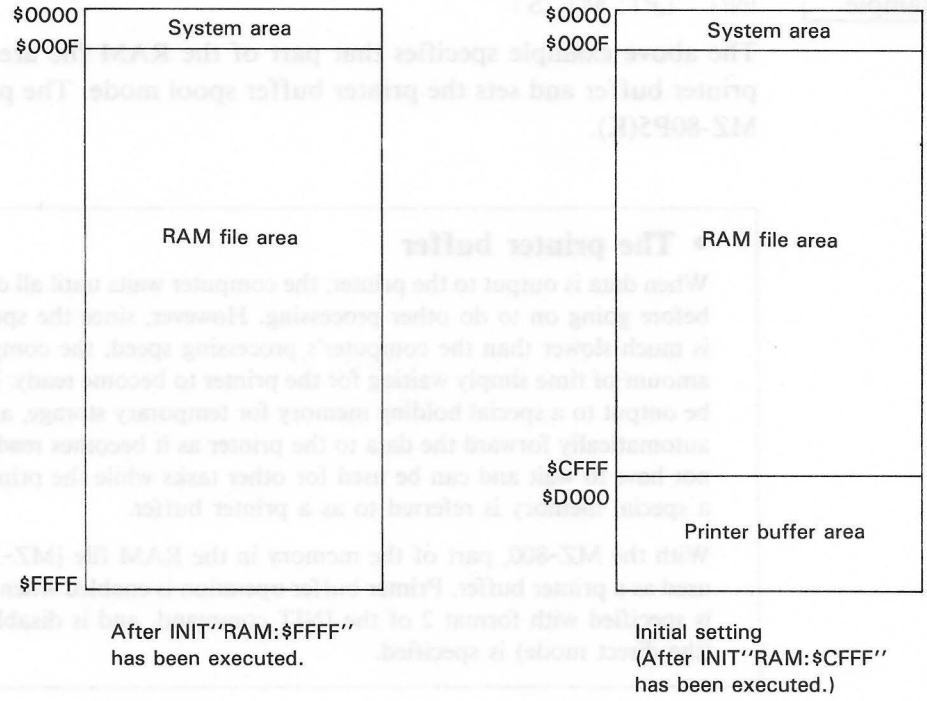
In this format the INIT command initializes the optional RAM file board (MZ-1R18) and allocates the amount of memory space specified in <number of bytes> to this file, with the remaining memory space reserved for the printer buffer. <number of bytes> must be within the range $0010 to $FFFF. When <number of bytes> is omitted, the current setting for the RAM file area is assumed. The "OK? [Y/N]" message appears when this command is executed. Typing Y sets up the RAM file area as shown below. Typing N causes BASIC to display a "Break" message and return to the command mode.

Either the RAM file board or the printer buffer function may become unavailable if the memory space assigned to it is too small.

Example

```
INIT "RAM:$FFFF"
```

This statement initializes the RAM file board and allocates the maximum amount of memory space to RAM files.



(Format 2)

In this format the INIT command specifies the printer and the mode in which the printer buffer is to be used.

[M] indicates the printer buffer mode.
M0: Direct mode (The buffer is initialized.)
M1: Spool mode (The buffer is initialized.)
M2: Direct mode (If the spool mode is active, this mode is entered after any existing contents of the buffer have been printed out.)
The M1 and M2 options are invalid if no RAM file board is installed.
An error will be generated if image print code 0BH + 0BH* is sent to the MZ-80P5(K) printer in the spool mode. To recover from this type of error, reenter the desired command after executing INIT "LPT: M2". Printing can be stopped in the spool mode by pressing the CTRL and N keys simultaneously.

* H indicates that the preceding number is in hexadecimal.

[S] specifies the printer type.
S0: MZ-1P16
S1: MZ-80P5(K)
S2: Printer which converts print data into ASCII codes
S3: Code through

The following codes are converted as shown during execution of PRINT/P statement when S0 or S1 is specified in the INIT statement.
CHR$ ($11) or ↓ is converted to $09.
CHR$ ($12) or ↑ is converted to $0B.
CHR$ ($15) or H is converted to $0F.
CHR$ ($16) or C is converted to $0C and $0A.

<CR code> must be specified when a code other than 0DH is to be used as the CR code.

Example

```
INIT "LPT: M1, S1"
```

The above example specifies that part of the RAM file area is to be used as the printer buffer and sets the printer buffer spool mode. The printer to be used is an MZ-80P5(K).

• The printer buffer

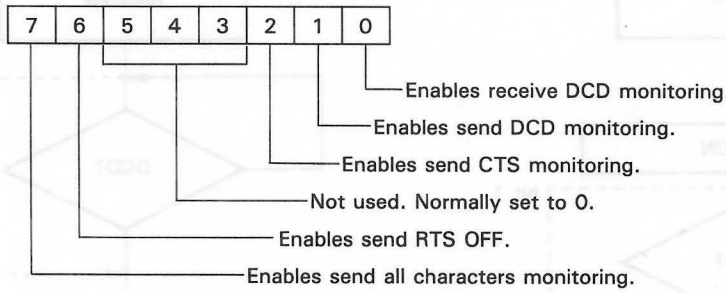
When data is output to the printer, the computer waits until all data has been printed before going on to do other processing. However, since the speed of data printout is much slower than the computer's processing speed, the computer spends a great amount of time simply waiting for the printer to become ready. However, if data can be output to a special holding memory for temporary storage, and that memory will automatically forward the data to the printer as it becomes ready, the computer will not have to wait and can be used for other tasks while the printer is printing. Such a special memory is referred to as a printer buffer.

With the MZ-800, part of the memory in the RAM file [MZ-1R18] option can be used as a printer buffer. Printer buffer operation is enabled when M1 (the spool mode is specified with format 2 of the INIT command, and is disabled when M0 or M2 (the direct mode) is specified.

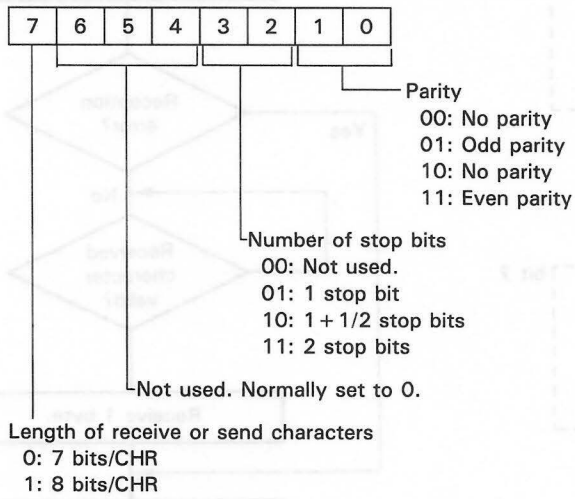
(Format 3)

In this format the INIT command sets up the RS-232C interface mode.

<Monitoring code> (High active)



<Initialization code> (High active)



<End code>

A number from 0 to 255 ($00 - $FF)

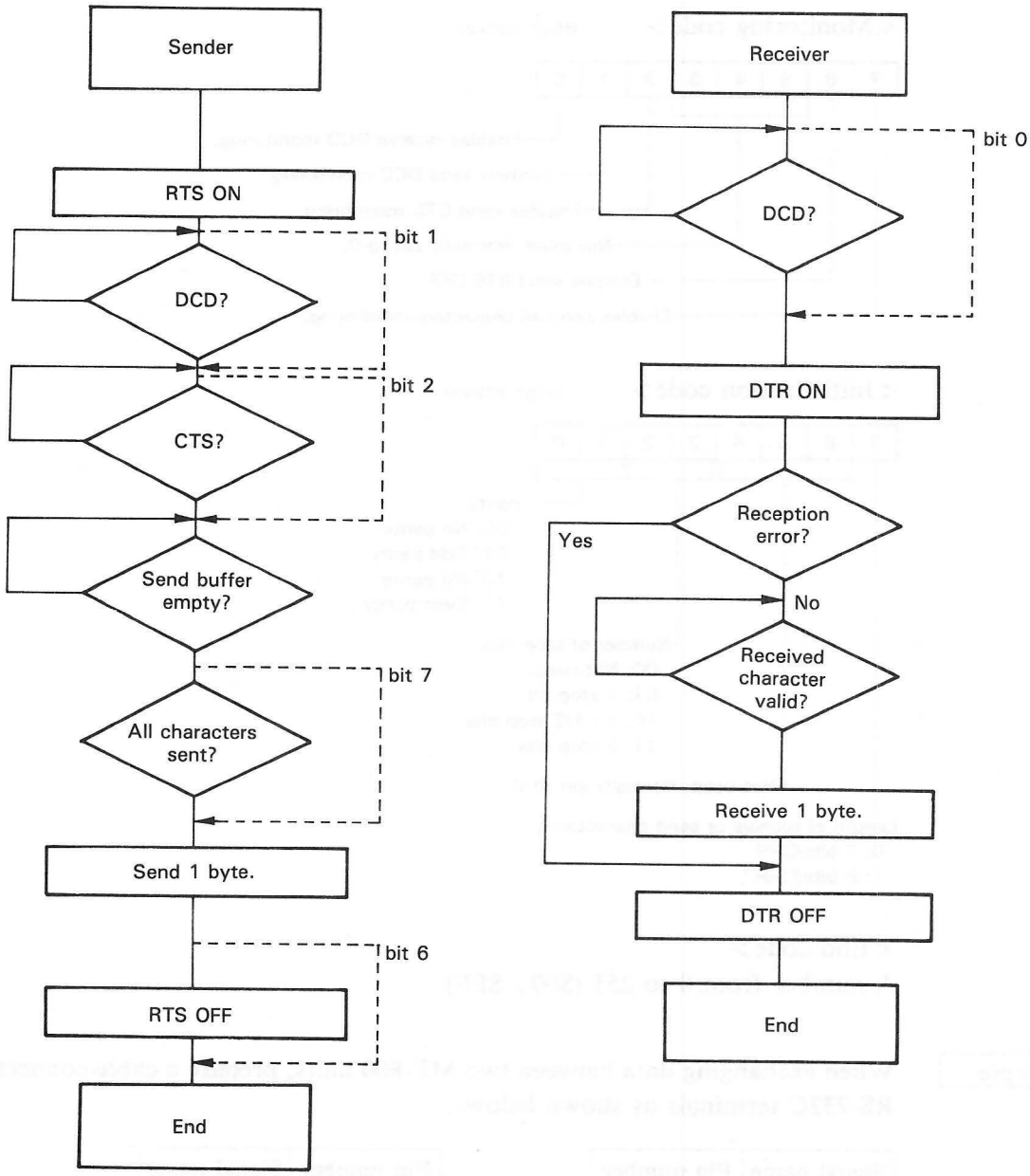
Example

When exchanging data between two MZ-800 units, prepare a cable connecting the RS-232C terminals as shown below.

Signal name	Pin number		Pin number	Signal name
TXD	2		2	TXD
RXD	3		3	RXD
RTS	4		4	RTS
CTS	5		5	CTS
DTR	6		6	DTR
DCD	7		7	DCD
Ground	1, 8		1, 8	Ground

Use both units in the terminal mode. (Refer to the manual for the RS-232C interface.)

<Flow chart>


Sender
RTS ON
bit 1
DCD?
bit 2
CTS?
Send buffer empty?
bit 7
All characters sent?
Send 1 byte.
bit 6
RTS OFF
End
Receiver
bit 0
DCD?
DTR ON
Reception error?
Yes
No
Received character valid?
Receive 1 byte.
DTR OFF
End

The following programs transfer the contents of A$ between the two MZ-800s:

[Program for sender]

```
10 INIT''RS1:$00,$8C''
20 A$=''0123456789''
30 WOPEN#1,''RS1:''
40 PRINT#1,A$
50 CLOSE#1
60 END
```

[Program for receiver]

```
10 INIT''RS1:$00,$8C''
20 ROPEN#2,''RS1:''
30 INPUT#2,A$
40 PRINT A$
50 CLOSE#2
60 END
```

(Format 4)

In this format the INIT command sets up the display settings. <mode> specifies the resolution of the screen and the number of colours as follows.

Mode	Resolution	Characters per line	Colours
1	320 × 200 dots	40	4 colours
2	320 × 200 dots		16 colours
3	640 × 200 dots	80	Foreground and background colours
4	640 × 200 dots		4 colours

Note:

Optional graphic memory (MZ-1R25) is required to set mode 2 or 4. When a TV set is used as the display unit, sufficient resolution will not be obtained in mode 3 or 4.

<block code> specifies the colour pallete block number.

See Appendix A for more information on display control.

BYE

Format **BYE**

Abbreviated Format

B.

Explanation The BYE command returns control of the computer from the BASIC interpreter to the monitor program in RAM.
See chapter 8 for details of the monitor program.

BOOT

Format **BOOT**

Explanation The BOOT command initiates an initial program load (IPL). This command places the computer into the same state as when the computer is first powered on.

Example BOOT ..... Reloads the system program into memory.

WAIT

Format **WAIT** <numeric data>

Abbreviated Format

W.

Explanation The WAIT statement suspends program execution for the time specified in <numeric data>. The time must be specified in milliseconds (1/1000 seconds).

Example WAIT 100 ..... Suspends program execution for 0.1 (100/1000) second.

6.3 File Control Statements

DIR

Format **DIR**[/P][RAM]

Explanation The DIR command displays the names of files on the RAM file board.
Specifying DIR/P sends the contents of the directory to the printer.
The optional MZ-1R18 RAM file board is required for this command to be valid.
RAM may be omitted when the RAM file board is specified in the DEFAULT statement or it is logged as the default device.
The device specified in the DIR command becomes the default device.
Each filename is followed by one of the following three file types.

BTX: BASIC program files
BSD: BASIC sequential data files or program files written in ASCII format
OBJ: Machine-language files

Example DIR RAM..... Displays a directory of the RAM file board files.

See also DEFAULT

RUN

Format **RUN** ["[<device name>:]<filename>"[,{A/R}]]

Abbreviated Format

R.

Explanation Erases the existing programs in the BASIC program area and clears the program work area, then loads the program specified with <filename> into the BASIC program area from the device indicated with <device name>. Then, this command executes the program from its beginning.
<device name> may be omitted when the default device or the device specified in the DEFAULT statement is to be used. When all parameters are omitted, this command does not erase the program in the BASIC program area.
To load and execute a program which has been saved in the form of BSD file written in ASCII codes, specify the A option.
Specifying the R option makes it possible to load an OBJ file in the same manner as IPL.
The file types which can be loaded are BTX, BSD and OBJ.

Example RUN "CMT:PROG" ..... Loads BTX file "PROG" from the cassette tape and executes it.

RUN "CMT:DATA",A ..... Loads BSD file "DATA" from the cassette tape and executes it.

LOAD

Format **LOAD** "[<device name>:]<filename>" [,A]

Abbreviated Format

LO.

Explanation The LOAD command loads a specified program into memory from an external storage device.

<filename> must have the same name as when the file was first saved. This parameter is mandatory. <device name> must be CMT or RAM. This parameter may be omitted when the default device or the device specified in the DEFAULT statement is to be used. Add the A option when loading a program file which is saved in ASCII format. Note that reading ASCII format files takes more time than binary format files.

Only BASIC text files and machine language programs can be loaded with this command. When the file to be loaded is a BASIC text file, the current program is cleared from the BASIC text area when the new program is loaded.

Note:

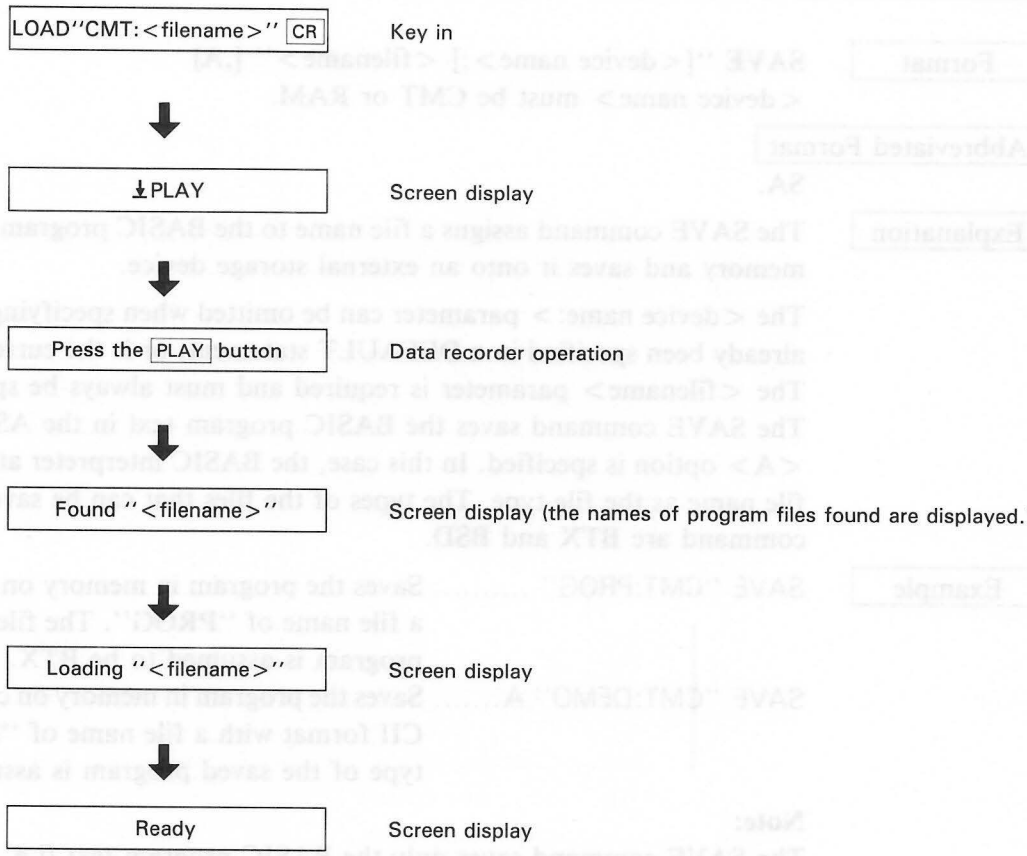
When loading a machine language routine to be linked with a BASIC program, the LIMIT statement must be executed to reserve an area in memory for the machine language program. Further, the applicable machine language program file is executed as soon as loading is completed if the loading address is inside that area. (In this case, the BASIC text is not erased.)

The LOAD command can be used within a program to load a machine language program file.

Example

```
LOAD "CMT:HELLO" ..... Loads a file named "HELLO" from the data
                       recorder.
```

Procedure for loading a program file



SAVE

Format

SAVE "[<device name>:] <filename>" [,A]
<device name> must be CMT or RAM.

Abbreviated Format

SA.

Explanation

The SAVE command assigns a file name to the BASIC program in the computer's memory and saves it onto an external storage device.

The <device name:> parameter can be omitted when specifying a device that has already been specified in a DEFAULT statement, or is the current default device. The <filename> parameter is required and must always be specified.
The SAVE command saves the BASIC program text in the ASCII format if the <A> option is specified. In this case, the BASIC interpreter attaches BSD to the file name as the file type. The types of the files that can be saved with the SAVE command are BTX and BSD.

Example

SAVE "CMT:PROG" ..... Saves the program in memory on cassette tape with a file name of "PROG". The file type of the saved program is assumed to be BTX.

SAVE "CMT:DEMO",A..... Saves the program in memory on cassette tape in ASCII format with a file name of "DEMO". The file type of the saved program is assumed to be BSD.

Note:
The SAVE command saves only the BASIC program text (i.e., the program text displayed by executing the LIST command); it does not save any machine language program in the machine language area.
When using SAVE, make a note of the tape counter reading for future reference.

Procedure for saving a program file

SAVE"CMT:<filename> [CR]	Key in
↓	
↓RECORD.PLAY	Screen display
↓	
Press the [RECORD] button.	Data recorder operation
↓	
Writing "<filename>"	Screen display
↓	
Ready	Screen display

VERIFY

Format **VERIFY** "[CMT]: <filename>"

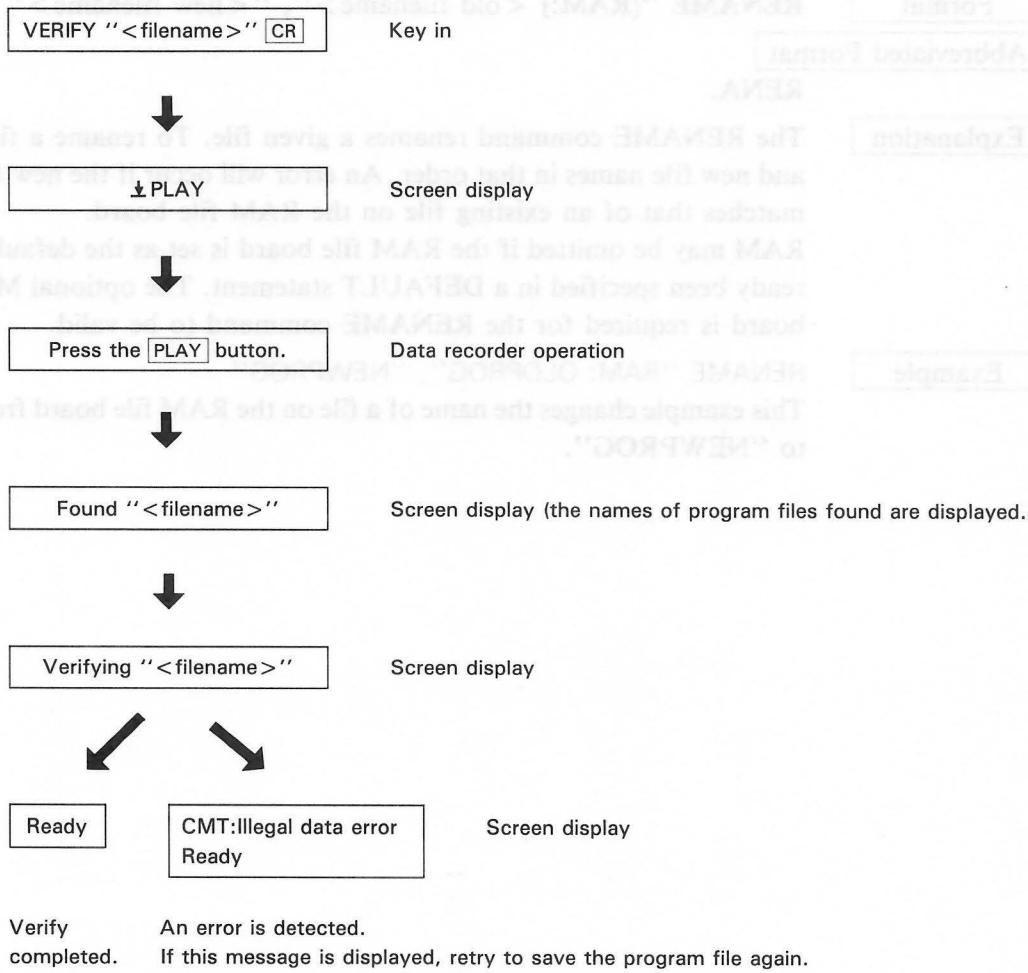
Abbreviated Format

V.

Explanation The VERIFY command compares the program in memory with the program written on cassette to confirm that the program has been properly saved. "Ready" is displayed if both programs are the same and "CMT:Illegal data error" is displayed if they are different. In the latter case, save the program again.
Any ASCII file cannot be verified.
This command is valid only for cassette files.

Example VERIFY "CMT:NAME"..... Compares file "NAME" on the cassette with the program in memory.

Procedure for verifying a program file



DELETE

Format **DELETE** "[RAM:] <filename>"

Abbreviated Format

D.

Explanation The DELETE command deletes the file specified in <filename>. The optional MZ-1R18 RAM file board is required for this command to be valid.

Example DELETE "RAM: SAMPLE"... Deletes a file named "SAMPLE" on the RAM file board.

RENAME

Format **RENAME** "[RAM:] <old filename>", "<new filename>"

Abbreviated Format

RENA.

Explanation The RENAME command renames a given file. To rename a file, specify the old and new file names in that order. An error will occur if the new file name specified matches that of an existing file on the RAM file board.
RAM may be omitted if the RAM file board is set as the default device or has already been specified in a DEFAULT statement. The optional MZ-1R18 RAM file board is required for the RENAME command to be valid.

Example RENAME "RAM: OLDPROG", "NEWPROG"
This example changes the name of a file on the RAM file board from "OLDPROG" to "NEWPROG".

CHAIN

Format

CHAIN "[<device name>:] <filename>"
<device name> must be CMT or RAM.

Abbreviated Format

CH.

Explanation

The CHAIN statement transfers execution from the current program to another program in a file. The CHAIN statement can also open a file. Executing a CHAIN statement has the same effect as executing the RUN command in a program except that CHAIN passes variables and arrays from the current program to the called program.

<device name> may be omitted when the default device or the device specified in the DEFAULT statement is to be used.

Example

```
10 A=1
20 B=2
30 CHAIN "CMT:PROG"
40 END
```

In this sample program, control is passed on line 30 to the program, from file "PROG" on the cassette. The values of variables A and B, 1 and 2, are passed to the called program.

MERGE

Format

MERGE ["[<device name>:] <filename>"][,A]
<device name> must be CMT or RAM.

Abbreviated Format

M.

Explanation

The MERGE command merges the program specified in the <filename> into the program currently in memory.
<device name> may be omitted when the default device or the device specified in the DEFAULT statement is to be used.
If lines from the file have the same line numbers as those in the program in memory, the lines from the file overwrite the corresponding lines in memory.
To merge a BSD file (program) saved in ASCII format, add the A option at the end of the statement.

Example

(Program in memory)

```
10 B=2
30 PRINT B
50 END
```

(Program on cassette tape)

```
"PROG"
10 A=1
20 PRINT A
40 END
```

When these programs are merged together with the MERGE "CMT: PROG" statement, the merged program will look like this:

```
10 A=1
20 PRINT A
30 PRINT B
40 END
50 END
```

Confirm the resulting program by using the LIST command.

WOPEN

Format

WOPEN # <logical number>, "[<device name>:] <filename>"
<logical number> must be an integer from 1 to 127.
<device name> must be CMT, RAM, or RSn.

Abbreviated Format

WO. #

Explanation

The WOPEN # statement opens a BSD file for output. It also assigns a logical number and name to the file.
<device name> may be omitted when the default device or the device specified in the DEFAULT statement is to be used. Specifying RSn as <device name> causes output to be sent to the RS-232C device.

Example

```
10 WOPEN #1, "CMT:DATA" ....
```
Opens a file under the name "DATA" for output and assigns logical number 1 to that file.

```
10 WOPEN #1, "RS1:" .....
```
After this statement is executed, all output from PRINT #1 statements is sent to the RS-232C port.

```
10 WOPEN #2, "DATA"
20 FOR Z=1 TO 99
30 PRINT #2, Z
40 NEXT Z
50 CLOSE #2
60 END
```

The above sample program writes numbers 1 to 99 into the specified file.

See also

PRINT #, ROPEN #, CLOSE #

PRINT

Format

PRINT # <logical number>, <data> [, <data>] ...

Abbreviated Format

? #

Explanation

The PRINT # statement writes data sequentially to the file that is opened for output with a WOPEN # statement.
<logical number> must be the file number used in the WOPEN # statement.
<data> may be numeric or alphanumeric.

Example

```
10 WOPEN #1, "CMT:DATA2"
20 PRINT #1, 1, 2, 3
30 CLOSE #1
40 END
```

This sample program writes numeric data 1, 2, and 3 into file "DATA". The file has the logical number 1 and is opened for output.

See also

WOPEN #, CLOSE #

ROPEN

Format

ROPEN # <logical number>, "[<device name>:] <filename>"
<logical number> must be an integer from 1 to 127.
<device name> must be CMT, RAM, or RSn.

Abbreviated Format

RO. #

Explanation

The ROPEN # statement opens a file for input. The ROPEN # statement assigns <logical number> to the file designated by <device name> and <filename>. <device name> may be omitted when the default device or the device specified in the DEFAULT statement is to be used. "RSn:" specified in <device name> designates the RS-232C interface as the input device from which data is to be read.

Example

10 ROPEN # 1, "CMT: DATA" .... Opens a BSD file named "DATA" on the cassette.

10 ROPEN # 1, "RS1:" ..... Sets the RS-232C port as the device from which all data specified in the INPUT # 1 statements is to be read.

```
10 ROPEN #2, "DATA"
20 FOR Z=1 TO 99
30 INPUT #2,A
40 PRINT A
50 NEXT Z
60 CLOSE #2
70 END
```

The above program reads and displays the contents of the file created by the sample program given for the WOPEN # statement.

See also

INPUT #, WOPEN #, CLOSE #

INPUT

Format **INPUT #** <logical number>, <variable> [, <variable>] ...

Abbreviated Format

I. #

Explanation The INPUT # statement sequentially reads data items from the file opened for input with the ROPEN # statement and assigns them to program variables. <variable> may be an array element. <logical number> is the same number used as when the file was first opened for input by the ROPEN # statement.
As with the READ ~ DATA statement pair, an error may be generated if data and variable types disagree. The end of file can be tested by using the EOF # function if the specified file is on the RAM file board.

Example

```
10 ROPEN #2, "DATA2"
20 INPUT #2, A, B, C
30 PRINT A, B, C
40 CLOSE #2
50 END
```

This sample program reads numeric data from the file opened for input under logical number 2 and assigns the data to numeric variables A, B, and C.

See also ROPEN #, CLOSE #, EOF #

EOF(#)

Format **EOF(** # <logical number>)

Abbreviated Format

EO. #

Explanation The EOF(#) function is used to find the end of a file. This function signals an end-of-file condition when all data in the file has been read. The value −1 (true) is returned after the end of the file is encountered. EOF(#) is invalid when reading data from CMT.
The EOF(#) function is generally used with the IF statement and placed after an INPUT # statement.

Example

```
10 ROPEN #3, "DATA"
20 INPUT #3, A
30 IF EOF(#3) THEN END
40 PRINT A
50 GOTO 20
```

The above program reads data items sequentially from the file named "DATA" and displays them on the screen until the end of the file is encountered.

See also INPUT #

CLOSE

Format **CLOSE**[# < logical number >]

Abbreviated Format

CLO. #

Explanation The CLOSE statement closes the file opened under the specified logical number. The logical number assigned to the file is released after execution of the CLOSE statement.

A CLOSE operation on a file opened for output causes the output buffer to be flushed. A CLOSE operation with no logical number specified closes all open files and releases all logical numbers.

Example

```
10 CLOSE#1 ..... Closes the file existing as logical number 1.
10 CLOSE ..... Closes all open files.
```

See also WOPEN#, ROPEN#

KILL

Format **KILL**[# <logical number>]

Abbreviated Format

KI. #

Explanation The KILL # command aborts the writing of data into the file opened under the specified logical number. A KILL # command with no logical number aborts all current writing processing, closes all open files, and releases the logical numbers.

Example

```
KILL#3 ..... Aborts the writing of data to the file opened under
             logical number 3 and releases the logical numbers as-
             signed to that file.
```

See also WOPEN#, PRINT#

DEFAULT

Format **DEFAULT** "<device name>: "

Abbreviated Format

DEF.

Explanation The DEFAULT statement defines the device names to be assumed when the <device name> parameter is omitted in input/output statements.

— Specify device names as follows:
CMT..... Data recorder (Default)
RAM..... RAM file board
LPT..... Printer
RS {1, 2}..... RS-232C interfaces

Example

```
DEFAULT "CMT: "
```

After this statement is executed, the data recorder becomes the default device whenever the <device name> parameter is omitted in input/output statements for external devices.

6.4 Graphics Control Statements

COLOR

Format **COLOR** [<palette code> [,<mode>]]

Abbreviated Format

COL.

Explanation The COLOR statement specifies the <palette code> and optional <mode> that are used by the PRINT, PRINT USING, and graphics statements SET, RESET, LINE, BLINE, BOX, CIRCLE, PAINT, PATTERN, and SYMBOL.
<mode> specifies the type of logical operation performed on the colours. When <mode> is specified as 0, the old colours in superimposed sections are over-painted by new colours. When this parameter is specified as 1, the old and new colours are logically ORed. The mode parameter does not apply however to the RESET and BLINE statements (see Appendix A).

Example

```
10 INIT "CRT:M1"
20 COLOR 3,0
30 FOR J=0 TO 10 STEP 2
40 SET 100,J
50 NEXT J
60 END
```

This program plots dots at points (100, 0) and (100,2) through (100,10) in colours associated with palette code 3 and in superimpose mode 0.

See also Appendix A.

PAL

Format

PAL <palette code>,<colour code>

Explanation

The PAL statement matches a palette code and colour codes to each other. Both the palette and colour code parameters can have a value from 0 to 15. In colour modes other than the 16-colour mode, the user can select two or four palette codes at a time and can select 16 colours. In the 16-colour mode, the user can set up a palette block with the INIT command and select four palettes for that block, again enabling selection of 16 colours. The default (initial) values of the palette and colour codes are given below.

(1) 2-colour mode

Palette code	Colour code
0	0 Black
1	15 Light white

(2) 4-colour mode

The table below shows the relationship between the palette and colour codes that is established when BASIC is started.

Palette code	Colour code
0	0 Black
1	1 Blue
2	2 Red
3	15 Light white

You can select four colour codes out of a possible 16 colour codes.

(3) 16-colour mode

The default palette code values are identical to those in colour mode. In this mode, colours are fixed for each palette block (see the "INIT Statement" for palette blocks).

n: Palette block number

n	Colour code	Colour
0	0	Black
	1	Blue
	2	Red
	3	Magenta
1	4	Green
	5	Cyan
	6	Yellow
	7	White

n	Colour code	Colour
2	8	Grey
	9	Light blue
	10	Light red
	11	Light magenta
3	12	Light green
	13	Light cyan
	14	Light yellow
	15	Light white

Note:

When a palette block is changed with the INIT statement in the 16-colour mode, the palette code settings are initialized. See Appendix A for details of colour codes and pallette codes.

SET

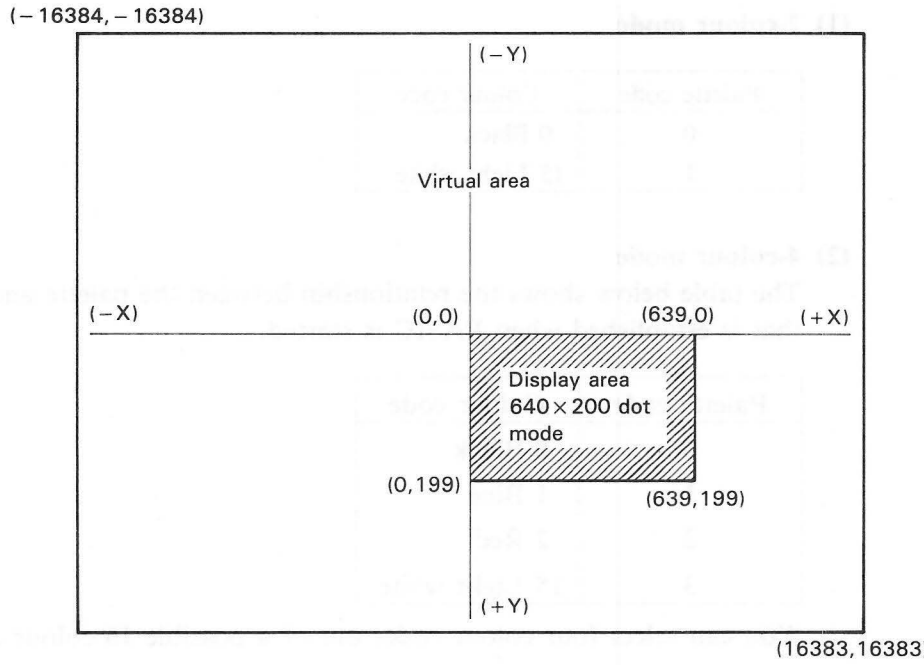
Format

SET [<colour specification>] <X-coordinate>,<Y-coordinate>
<colour specification> = [<palette code>][,<mode>]

Explanation

The SET statement sets a dot on the screen at the point specified by <X-coordinate> and <Y-coordinate> in the specified colour. <X-coordinate> and <Y-coordinate> are numerical expressions (i.e., numeric constants, variables, or expressions). They can have values from the following ranges:

$-16384 \leqq$ <X-coordinate> $\leqq 16383$
$-16384 \leqq$ <Y-coordinate> $\leqq 16383$



Although you can specify X- and Y-coordinates in the virtual area, BASIC displays only the shaded area in the above figure. <palette code> can specify the colour of the dot to be plotted. <mode> must be either 0 or 1. When 0 is specified, the dot is displayed in the colour specified by <palette code>, irrespective of the current palette code value. When 1 is specified, the dot is displayed in the colour determined by ORing the current palette code with the <palette code> specified in the SET statement.

When <colour specification> is omitted, the dot is displayed in the colour specified by the last COLOR statement.

Example

```
10 SET[3,0] 100,50.....
```

Turns on a dot at coordinates (100,50) in the colour associated with the palette code 3, superimpose mode 0.

See also

RESET

RESET

Format

RESET [<colour specification>] <X-coordinate>,<Y-coordinate>
<colour specification> = <palette code>,<superimpose mode>

Explanation

The RESET statement changes the colour of a dot on the screen at the point specified by <X-coordinate> and <Y-coordinate> according to the rule shown below. <X-coordinate> and <Y-coordinate> are numerical expressions (i.e., numeric constants, variables, or expressions). They can have values in the following ranges:

$-16384 \leqq$ <X-coordinate> $\leqq 16383$
$-16384 \leqq$ <Y-coordinate> $\leqq 16383$

Their range of values is the same as that for the SET statement. <palette code> specifies the palette code for the colour of the dot to be reset. <mode> must be either 0 or 1. See Appendix A for more information.

See also

SET

LINE

Format

LINE [<colour specification>] <X-coordinate>, <Y-coordinate>, <X-coordinate>, <Y-coordinate> [,<X-coordinate>, <Y-coordinate>] ...
<colour specification> = [<palette code>][,<mode>]

Explanation

The LINE statement draws line(s) connecting given points in the specified colour. <X-coordinate> and <Y-coordinate> are numerical expressions (i.e., numeric constants, variables, or expressions). Their range of values is the same as that for the SET statement. The <colour specification> parameter is identical to that of the SET statement. If this parameter is omitted, the colour specification made in the COLOR statement is assumed. If coordinates outside the display area are specified, the line is clipped off at the boundary of the display area.

Example

```
10 LINE [2,0]10,20,260,180,380,60
20 END
```

The above program draws lines that connect from points (10,20), (260,180), to (380, 60) in the colour previously specified from palette code 2 in superimpose mode 0.

```
10 INIT''CRT:M1''
20 FOR X1=0 TO 319 STEP 3
30 LINE 159,99,X1,0
40 NEXT X1
50 FOR Y1=0 TO 199 STEP 3
60 LINE 159,99,319,Y1
70 NEXT Y1
80 FOR X2=319 TO 0 STEP -3
90 LINE 159,99,X2,199
100 NEXT X2
110 FOR Y2=199 TO 0 STEP -3
120 LINE 159,99,0,Y2
130 NEXT Y2
140 END
```

The above program draws dotted lines (every three dots) from the center of the screen (159,99) to the corners of the screen.

See also

BLINE, SET

BLINE

Format

BLINE [<colour specification>] <X-coordinate>, <Y-coordinate>, <X-coordinate>, <Y-coordinate> [,<X-coordinate>, <Y-coordinate>]...
<colour specification> = <palette code>,<superimpose mode>

Explanation

The BLINE statement changes the colour of line(s) connecting given points on the screen according to the rule shown below. <X-coordinate> and <Y-coordinate> are numerical expressions (i.e., numeric constants, variables, or expressions). Their range of values is the same as that for the SET statement. The <colour specification> parameter is identical to that of the RESET statement. If this parameter is omitted, the colour specification made in the COLOR statement is assumed. If coordinates outside the display area are specified, only the line segment within the display area is deleted. See Appendix A for more information.

See also

LINE, RESET

BOX

Format

BOX [<colour specification>] <X-coordinate 1>, <Y-coordinate 1>, <X-coordinate 2>, <Y-coordinate 2> [,<palette code>]
<colour specification> = [<palette code>][,<superimpose mode>]

Explanation

The BOX statement uses two pairs of coordinates as the location of the opposing corners of the box. <X-coordinate> and <Y-coordinate> are numerical expressions. Their range of values is the same as that for the SET statement.
The <colour specification> parameter is identical to that of the SET statement. If this parameter is omitted, the colour specification mode in the COLOR statement is assumed.
The last <palette code> parameter specifies that the box must be painted in the specified colour. When this parameter is omitted, only the borders are drawn.

Example

```
10 INIT''CRT:M1''
20 CLS
30 BOX [2,0]20,20,60,60,2
40 END
```

This program draws a rectangle on the screen and paints it in colour previously specified from palette 2.

See also

SET

CIRCLE

Format

CIRCLE [<colour specification>] <X-coordinate>, <Y-coordinate>, <radius> [,[<aspect>] [,<start>,<end>][,O]]
<colour specification> = [<palette code>][,<superimpose mode>]

Abbreviated Format

CI.

Explanation

The CIRCLE statement draws an ellipse (circle) or arc (fan). The meanings of the <colour specification> parameter are identical to those of the SET statement. When this parameter is omitted, the values specified by the COLOR statement are assumed. <X-coordinate> and <Y-coordinate> give the coordinates of the center of the circle and <radius> the radius of the circle. Their ranges of values are as follows:

$-16384 \leqq$ <coordinates> $\leqq 16383$
$0 \leqq$ <radius> $\leqq 16383$

The area in which the circle can be actually displayed is determined by the INIT command.
<aspect> affects the ratio of the X-radius to the Y-radius. When <aspect> is less than 1, the <radius> specified becomes the X-radius. If aspect is greater than 1, then <radius> becomes the Y-radius. The default value of <aspect> is 1.
The <start> and <end> angle parameters specify where drawing of an ellipse is to begin and end. These parameters must be given in radians. When omitted, an ellipse (circle) is drawn. When the O parameter is specified with <start> and <end>, a fan is drawn, that is, an arc is connected to the center point with lines. When O is omitted, an arc only is drawn.

Example

```
10 INIT''CRT:M1''
20 CIRCLE[1,0]100,100,80,0.5
30 GOSUB 80
40 CIRCLE[2,0]50,130,60,0.5,0,π/4,O
50 GOSUB 80
60 CIRCLE 159,99,50
70 END
80 GET A$:IF A$='' '' THEN 80
90 RETURN
```

The above program draws an ellipse, and, if any key is pressed, it draws an arc, then a circle.

See also

SET, GET

PAINT

Format

PAINT [<palette code>] <X-coordinate>, <Y-coordinate>, <boundary colour> [,<boundary colour>] ...

Explanation

The PAINT statement fills in an area on the screen with the colour specified by <palette code>.

When <palette code> is omitted, the palette specified in the COLOR statement is assumed.

You can select the <boundary colour> from 16 colours. The range of values that <X-coordinate> and <Y-coordinate> can have is determined by the INIT statement.

Unless the area is completely surrounded by the specified border colour (called the closed loop state), painting will occur beyond that area. Painting will be suppressed if the specified X- and Y-coordinates lie on the border or in an area that has already been painted with the specified colour.

Figures are all drawn in dots, so when lines and curves are drawn in a small area, small closed loops may result. When this happens, painting will not occur unless coordinates falling inside the closed loop are specified.

Example

```
10 INIT''CRT:M1''
20 CLS
30 CIRCLE[2]160,100,50
40 PAINT[1]160,100,2
50 END
```

The above program paints the area surrounded by a border using palette code 2 with a colour specified by palette code 1, starting at point (160,100).

PATTERN

Format

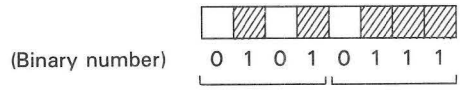
PATTERN [<colour specification>]<numeric data>, <text data>
<colour specification> = [<palette code>][,<superimpose mode>]

Explanation

The PATTERN statement defines a graphics pattern in the specified colour. The meanings of the <colour specification> parameter are identical to those of the SET statement. When this parameter is omitted, the parameters in the COLOR statement are assumed.

The pattern to be drawn can be specified using <numeric data> and <text data>. <numeric data> (±1 to ±24) represents the number of stacked 8-bit dot pattern rows, and <text data> represents the individual dot pattern rows.

Drawing of the pattern is controlled by the position pointer. The number of dot pattern rows specified by <numeric data> are displayed from bottom to top if <numeric data> is positive and from top to bottom if it is negative. After the specified number of dot pattern rows are drawn, dot pattern lines 8 bits (1 character) to the right of the current column are displayed. <text data> must be specified using ASCII codes which correspond to the binary representation of dot pattern rows.

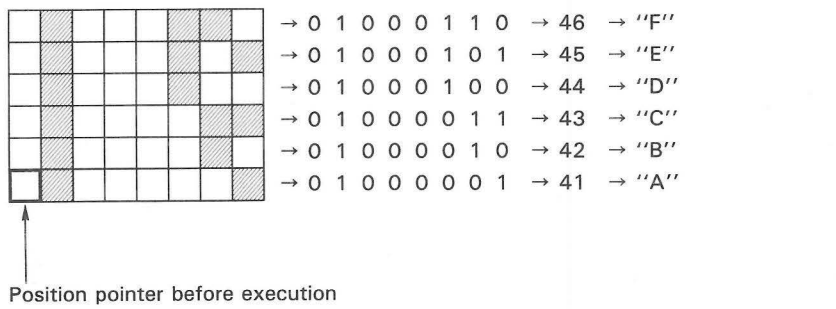


→ 57 (hexadecimal)
→ CHR$($57) or "W"

Example

```
10 POSITION 100,100..... Sets up the position pointer.
20 PATTERN[2,0]6,"ABCDEF"
30 END
```

The above program draws the graphics pattern shown below in a colour from palette 2.



Line 20 above can be replaced by the following line:

```
20 PATTERN[2,0]6,CHR$($41,$42,$43,$44,$45,$46)
```

See also

POSITION

POSITION

Format **POSITION** <X-coordinate>,<Y-coordinate>

Abbreviated Format

POS.

Explanation The POSITION statement sets the position pointer to a given point on the screen. The position pointer points to the position on the screen where the dot pattern specified in a subsequent PATTERN statement is to be displayed. The range of values of the <X-coordinate> and <Y-coordinate> parameters is the same as that of the SET statement.

Example

```
10 POSITION 100,50
20 A$="ABCDEFGH"
30 PATTERN[1,0]-8,A$
40 END
```

The POSITION statement on line 10 sets the position pointer to (100,50) where execution of the subsequent PATTERN statement begins.

See also PATTERN

SYMBOL

Format

SYMBOL [<colour specification>] <X-coordinate>,<Y-coordinate>,
<text data>[,<horizontal magnification>]*[,<vertical magnification>]*
[,<angle code>]
<colour specification> = [<palette code>][,<superimpose mode>]

Abbreviated Format

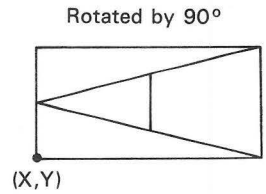
SY.

Explanation

The SYMBOL statement draws a graphics pattern of a given size at a given angle. When this statement is encountered, BASIC positions the lower left corner of the graphics pattern represented by <text data> at point (X-coordinate, Y-coordinate), rotated by <angle code>, and magnified by a factor of <horizontal magnification> and/or <vertical magnification>.
The meanings of the <colour specification> parameter are identical to those of the SET statement. When this parameter is omitted, the parameters in the COLOR statement are assumed.

The range of values that <X-coordinate> and <Y-coordinate> can have is identical to that specified in the SET statement. <horizontal magnification> and <vertical magnification> are integers from 1 to 255 and default to 1.
When <angle code> is specified, the pattern is rotated counterclockwise with respect to the upper left corner of the pattern at point (X-coordinate, Y-coordinate) by the angle specified by <angle code>. The reference position of the pattern remains unchanged after the rotation. The relationship between angle codes and angles is given below.

Angle code	Angle
0	0° (Default)
1	90°
2	180°
3	270°



Example

```
10 SYMBOL [1] 40,0, ''MZ-800'', 5,5,0
20 FOR J=1 TO 3
30 SYMBOL [J] J*80,100, ''A'',J+2,J+2,J
40 NEXT J
50 SYMBOL [1] 280,199, ''MZ-800'', 5,5,2
```

POINT

Format

POINT (<X-coordinate>,<Y-coordinate>)

Explanation

The POINT function returns the palette code that is defined at the given point on the screen. The range of values that <X-coordinate> and <Y-coordinate> can have is set by the INIT statement.

Example

```
10 INIT''CRT:M1''
20 SET[3,0]100,100
30 A=POINT(100,100)
40 PRINT A
50 END
RUN
3
Ready
```

The statement on line 10 assigns the point (100,100) to palette code 3. The POINT function therefore returns a palette code of 3.

6.5 Music Control Statements

MUSIC

Format

MUSIC <note1 of melody1>[;<note1 of melody2>]
[;<note1 of melody3>][,<note2 of melody1>]
[;<note2 of melody2>][;<note2 of melody3>] ...

Abbreviated Format

MU.

Explanation

The MUSIC statement generates through the MZ-800 speaker the melody or sound effects specified by the character string or string variable of its argument.
Three parts of a melody can be played at the same time. In the melody specification that follows the keyword MUSIC, where a melody is a sequence of notes, semicolons are used to separate individual parts, and commas are used to separate one melody from another. Each note is specified as follows:

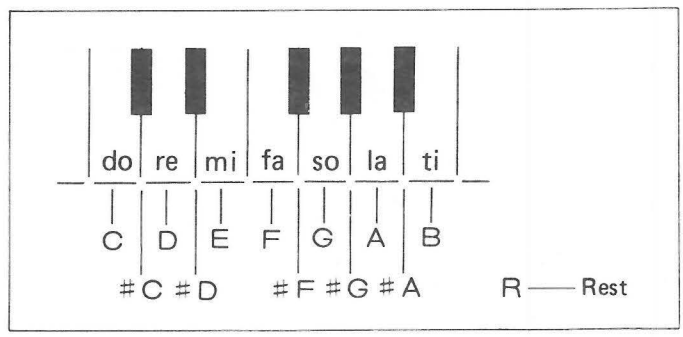
<octave specification> <note name> <duration>

(i) Octave specification
The basic octave of a melody is specified in the format "On" where n is a number in the range 0 to 6. The plus sign (+) makes the following notes played one octave higher than the basic octave, while the minus sign (−) causes notes to be played one octave lower. If neither sign is specified, the basic octave is assumed. Plus and minus signs are illegal in a melody in which the basic octave is set to O6 and O0, respectively.

(ii) Note specification
The symbols used to specify notes within each octave are as follows:

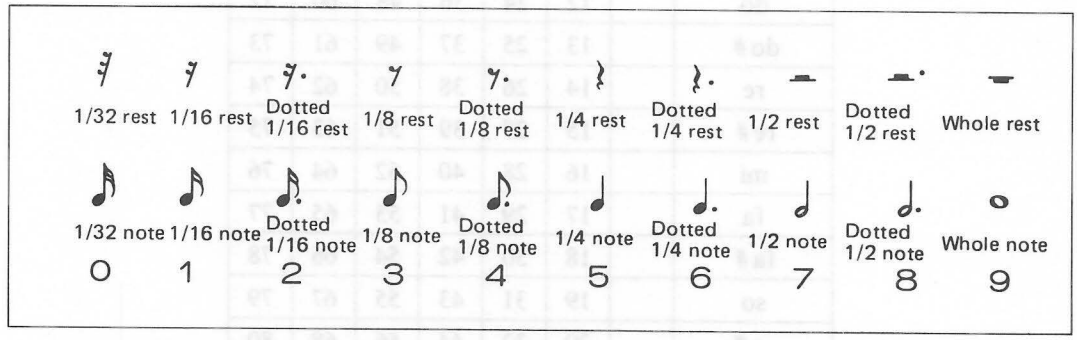
CDEFGAB#R

The relationship between the 8-note scale (do, re, mi, fa, so, la, ti, do) and these symbols are shown below. The sharp symbol (#) is used to raise a note by a half step. A note can be lowered a half step by attaching a # symbol to a note one step lower than that desired. For example, B flat is represented as #A. Silent intervals are specified with "R".



(iii) Duration specification

The duration specification determines the length of the specified note. Durations from 1/32 to 1 are specified as numbers from 0 to 9 as shown below. (When R is specified, the length of the silent interval is determined.)



Example

The following program plays "Oh! Susanna", composed by Stephen Foster:

```
10 TEMPO 6
20 A1$="O2G1A1B3+D3+D3+E3+D3B3G4A1B3B3A3G3A6G1A1B3+
   D3+D3+E3+D3B3G4A1B3B3A3A3G6R3"
30 A2$="G1A1G3B3B3+C3B3G3G4G1G3G3G3G3E6G1A1G3B3B3+
   C3B3G3G4G1G3G3E3E3G6R3"
40 MUSIC A1$;A2$
50 END
```

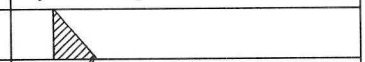
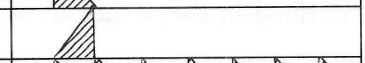
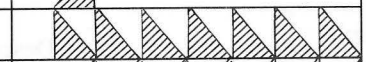
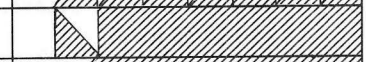
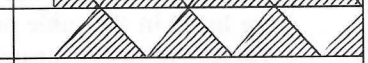
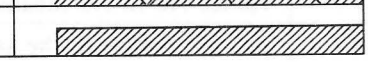
The following options can be defined in the <melody> specification:

	n = integer value	Default value	Description
On	0 to 6	2	Sets the current octave. The frequency of A is 440Hz if n = 2.
Nn	0 to 83		Specifies the note directly through the corresponding note number rather than through the octave number and note name. The values of n are listed in the table on the following page. Values N0 to N8 means rest.
Tn	1 to 7	4	Sets the tempo in the same way as the TEMPO statement.
Vn	0 to 15	15	Sets the sound volume. The volume is maximum when n = 15 and no sound is generated when n = 0.
Sn	0 to 7	8	Sets the envelope pattern (sound waveform). For the values of n, see the figures for envelopes on the following page.
Mn	1 to 255 1 = approx. 10m/s	255	Always used with the S parameter to specify the rate at which the envelope pattern is to change. The rate is maximum when n = 1 and decreases. The slope of the envelope becomes slower as the value of n increases. Values of Mn which are too large may generate inaudible sound depending on the envelope pattern.
Ln	0 to 9	5	Sets the length of the note.

Note numbers (Nn)

Note \ Octave	0	1	2	3	4	5	6
do		12	24	36	48	60	72
do #		13	25	37	49	61	73
re		14	26	38	50	62	74
re #		15	27	39	51	63	75
mi		16	28	40	52	64	76
fa		17	29	41	53	65	77
fa #		18	30	42	54	66	78
so		19	31	43	55	67	79
so #		20	32	44	56	68	80
la	9	21	33	45	57	69	81
la #	10	22	34	46	58	70	82
ti	11	23	35	47	59	71	83

Envelope patterns (Sn)

n	Envelope pattern (x-axis represents time; y-axis represents volume.)
0	
1	
2	
3	
4	
5	
6	
7	
8	

x-axis represents time in the units specified in Mn.

The MUSIC statement causes music data to be buffered and the sounds to be generated independently of computer processing. This makes it possible to change the display on the screen while playing music. However, this function also prevents the music from being stopped with the [SHIFT] and [BREAK] keys or being played out to the display. The following commands are provided to control the starting and stopping of musical sound:

MUSIC STOP: Stops sound generation.

MUSIC WAIT: Suspends program execution until the entire series of notes have been played.

MUSIC INIT: Initializes the music and noise setting to "O2V15L5T4S8M255". (See the table on the previous page.)

TEMPO

Format

TEMPO <numeric expression>

Abbreviated Format

TE.

Explanation

The TEMPO statement sets the tempo with which music is played by the MUSIC statement. The setting for tempo may range from 1 to 7. The default setting is 4.

TEMPO 1: Slowest tempo,
TEMPO 4: Medium tempo,
TEMPO 7: Fastest tempo

SOUND

Format

SOUND <pitch>,<duration>
SOUND=(<register>,<data>)

Abbreviated Format

SO.

Explanation

The SOUND statement generates sounds as specified by <pitch> and <duration>.

(i) <pitch>

<pitch> specifies the pitch of the sound. The pitch codes and the corresponding musical notes are listed in the table below.

Octave / Note	0	1	2	3	4	5	6
do		12	24	36	48	60	72
do #		13	25	37	49	61	73
re		14	26	38	50	62	74
re #		15	27	39	51	63	75
mi		16	28	40	52	64	76
fa		17	29	41	53	65	77
fa #		18	30	42	54	66	78
so		19	31	43	55	67	79
so #		20	32	44	56	68	80
la	9	21	33	45	57	69	81
la #	10	22	34	46	58	70	82
ti	11	23	35	47	59	71	83

The frequency of note la in octave 2 is 440 Hz.

(ii) <duration>

<duration> specifies in units of 1/100 seconds the length of the tone generated by this statement. <duration> must be a numeric expression from 0 to 65535. The SOUND = (<register>,<data>) statement is used to directly control the sound generator (Programmable Sound Generator) LSI. The PSG can generate three tones and one noise. The PSG register table is shown below.

Regiser Number			Data
0	Tone 0	Frequency	Integer from 1 to $2^{10} - 1$
1	Tone 0	Volume	Integer from 0 to 15 (see note)
2	Tone 1	Frequency	Same as register 0
3	Tone 1	Volume	Same as register 1
4	Tone 2	Frequency	Same as register 0
5	Tone 2	Volume	Same as register 1
6	Noise	Frequency	Noise data
7	Noise	Volume	Same as register 1

The PSG can generate either synchronous or white noise. The type of noise to be generated can be specified by sending 1-byte of data to PSG port $F2. See Appendix B for control of PSG.

NOISE

Format

NOISE <melody>[,<melody>]...

<melody> = $\left\{\begin{matrix}+\\-\end{matrix}\right\}$ <note name>[<duration>]

Abbreviated Format

NO.

Explanation

The NOISE statement generates white noise as specified by the <melody n> parameters. The meanings of the <melody n> parameters are identical to those of the MUSIC statement.

This statement can generate two parts of noises simultaneously. The parameters specifying these two must be separated by a semicolon (;). Any two consecutive melodies must be separated by a comma (,).

Example

```
10 NOISE "C3D1","E3F1"
```

The above NOISE statement generates two parts of white noises simultaneously.

6.6 Printer Control Statements

PTEST

Format **PTEST**

Abbreviated Format

PTE.

Explanation The PTEST command causes the printer to print squares in black, blue, green, and red in that order to check the colour specification, quantity of pen ink, and so on.



0 (Black) 1 (Blue) 2 (Green) 3 (Red) ← Value specified in PCOLOR

This command is valid only in the text mode.

PMODE

Format **PMODE** { TN / TL / TS / GR }

Abbreviated Format

PM.

Explanation The PMODE command specifies the operating mode for the colour plotter-printer.

PMODE TN

The PMODE TN command returns the printer to the text mode from the graphics mode, and sets the character size to 40 characters per line (the initial setting).

PMODE TL

The PMODE TL command returns the printer to the text mode from the graphic mode, and sets the character size to 26 characters per line.

PMODE TS

The PMODE TS command returns the printer to the text mode from the graphic mode, and sets the character size to 80 characters per line.

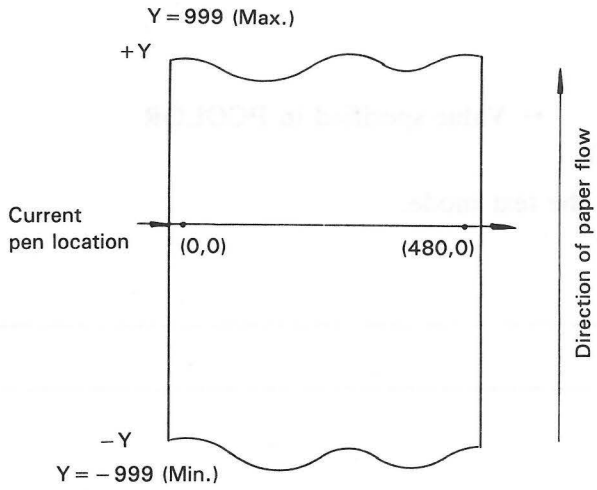
PMODE GR

The PMODE GR command switches the printer from the text mode to the graphics mode. When switching to this mode, the BASIC program being executed must make a note of the character size being used immediately before the mode change is made. Doing this allows the program to return to the text mode when the SHIFT + BREAK key is pressed or a STOP command is encountered.

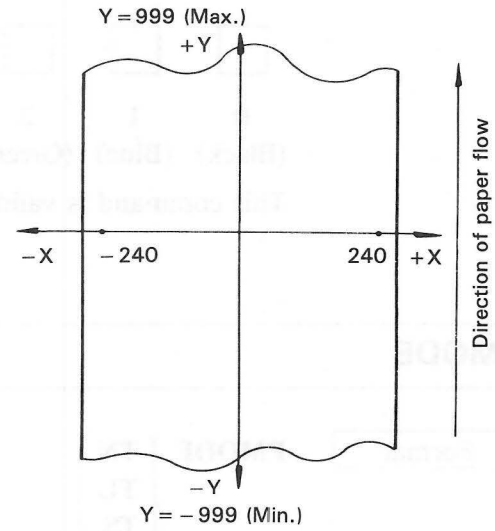
*** CHARACTER MODE ***

SHARP MZ-800	80 character mode (TL)
SHARP MZ-800	40 character mode (TN)
SHARP MZ-800	26 character mode (TS)

The PMODE GR command turns any command used in the graphics mode executable and sets the X and Y axes on the printer. The current pen location is initially set as the origin (0, 0). The origin can be moved to any location if it is within the range of the printable area. Printing beyond the forms may damage the pen and cause printer trouble.



X and Y axes after the origin is moved to the center. (PMOVE 240, -240: HSET)



X and Y axes after execution of a PMODE GR command. The X-axis is drawn from 0 to 480 and the Y-axis from -999 to 999.

• Printer modes

The modes of printer operation and commands which can be used with different modes are as shown in the table below.

Mode	Mode selection command	Commands usable	
Text mode 40 characters/line	PMODE TN	PTEST PCOLOR PSKIP PAGE * PRINT/P * PRINT/P USING	* LIST/P * HCOPY PLOT
Text mode 26 characters/line	PMODE TL		
Text mode 80 characters/line	PMODE TS		
Graphic mode	PMODE GR	PLINE RLINE PMOVE RMOVE PHOME	HSET GPRINT AXIS PCIRCLE PCOLOR

Note:
Commands marked with an asterisk (*) can be used with a dot printer (MZ-80 P5(K)); other commands can only be used with a plotter printer.

PCOLOR

Format **PCOLOR** {0 / 1 / 2 / 3}

Abbreviated Format

PC.

The PCOLOR command specifies the colour to be used for the printout of characters or graphics. This command can be entered in either the text mode or graphics mode. The available colours and the corresponding colour numbers are listed below.

0: Black
1: Blue
2: Green
3: Red

Example

10 PCOLOR 1 ..... Sets the pen colour to blue.

PSKIP

Format **PSKIP** <number of lines>

Abbreviated Format

PS.

Explanation

The PSKIP command feeds the paper by the specified <number of lines> forward when the specified value is positive and feeds it by the specified <number of lines> backward when the value is negative. <number of lines> must be an integer from −20 to 20. This statement is valid only in the text mode.

Example

10 PSKIP 12..... Feeds the paper 12 lines forward.
20 PSKIP −6..... Feeds the paper 6 lines backward.

PAGE

Format **PAGE** <number of lines>

Abbreviated Format

PA.

Explanation

The PAGE command specifies the number of lines per page. <number of lines> must be an integer from 1 to 72. The PAGE command also sets the current page position as the first line of the page. This statement can only be executed in the text mode.

Example

10 PAGE 20 ..... Sets the number of lines per page to 20. With this setting, the printer will space 20 lines when a form feed is performed.

PRINT/P

Format **PRINT/P** <data> [{;,}<data>] ...

Abbreviated Format

?/P

Explanation The PRINT/P statement submits output data to the printer in almost the same format as the PRINT statement would to the screen. Either the separators "," and ";" or the TAB function in this statement have the same effect as that in the PRINT statement. Various functions supported by the printer can be used by sending print control codes in the following format:

PRINT/P CHR$ (control code)

Example

10 PRINT/P "ABCD"..... Prints "ABCD".
10 PRINT/P CHR$($0A) ..... Causes a line feed.

Note:

To execute PRINT/P statements containing control code ↑ or ↓ successively, you must specify them on separate lines or delimit them with colons (:). BASIC may interpret control codes concatenated with connectors (+) as a single code sequence and cause a print malfunction.

[Invalid] PRINT/P "↓" + "↓ ↓ 4D"

BASIC will interpret this statement as PRINT/P "↓ ↓ ↓" +"4D". Instead, specify the following:

[Valid] PRINT/P "↓" : PRINT/P "↓ ↓ 4D"

PRINT/P USING

Format **PRINT/P** [<palette code>]**USING**"format string";<data> [{;,}<data>]...

Abbreviated Format

?/P USI.

Explanation The same as the PRINT USING statement excepting that the output device is the printer.

See also PRINT USING

PLINE

Format

PLINE [% <line type>,] x1,y1 [, x2,y2, ..., xi,yi]

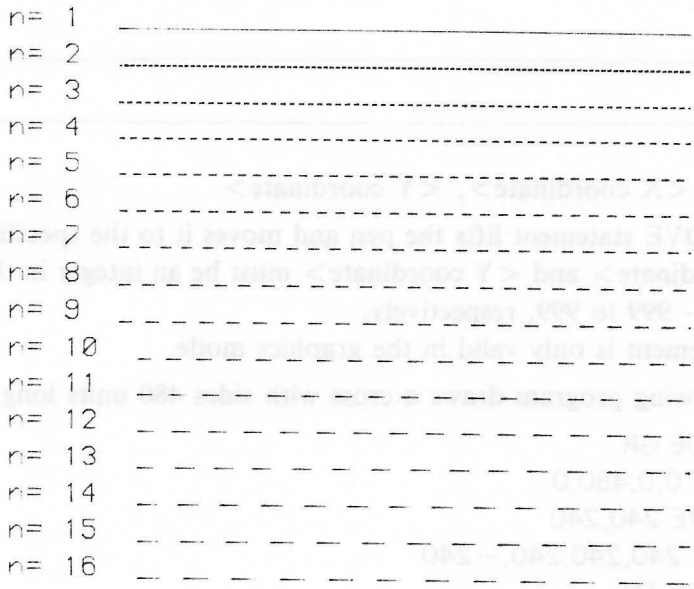
Abbreviated Format

PLI.

Explanation

The PLINE statement draws a solid or dotted line from the current pen location to the location indicated by absolute coordinates (x1,y1), then draws a line from that point to the location indicated by absolute coordinates (x2,y2), etc. xi must be an integer from −480 to 480 and yi an integer from −999 to 999. <line type> specifies the type of line to be drawn and must be an integer from 1 to 16. Solid lines are drawn when <line type> = 1 and dotted lines are drawn when <line type> = 2 to 16, where n is a number corresponding to a line type. If % <line type> is omitted, the previous value of n is assumed. The initial value of <line type> is 1 (solid line). Lines selectable with <line type> are as follows:

* * * LINE 1—16 * * *



The PLINE statement is only valid in the graphics mode.

Example

```
10 PMODE GR
20 PLINE %1, 0, 0, 200, 0, 200, −200, 0, −200, 0, 0
30 END
```

The above program draws a square with sides 200 units long.

RLINE

Format

RLINE [%<line type>,] x1,y1 [, x2,y2, ..., xi, yi]

Explanation

The RLINE statement draws a line from the current pen location to the location indicated by relative coordinates (x1,y1), then draws a line from that point to the location indicated by relative coordinates (x2,y2), etc.
xi must be an integer from −480 to 480 and yi must be an integer from −999 to 999. The line styles selectable with <line type> are the same as for the PLINE statement.
The RLINE statement is only valid in the graphics mode.

Example

```
10 PMODE GR
20 SQ=INT (120*SQR(3))
30 RLINE %1,240,0,-120,-SQ,-120,SQ
40 PMODE TN
```

This program draws a triangle with solid lines.

PMOVE

Format

PMOVE <X coordinate>, <Y coordinate>

Explanation

The PMOVE statement lifts the pen and moves it to the specified location (x,y). <X coordinate> and <Y coordinate> must be an integer in the range −480 to 480 and −999 to 999, respectively.
This statement is only valid in the graphics mode.

Example

The following program draws a cross with sides 480 units long:

```
10 PMODE GR
20 PLINE 0,0,480,0
30 PMOVE 240,240
40 PLINE 240,240,240,-240
50 PMODE TN
```

Remember to advance the paper before executing this program.

RMOVE

Format **RMOVE** <X coordinate>,<Y coordinate>

Abbreviated Format

RM.

Explanation The RMOVE statement lifts the pen and moves it to the location indicated by relative coordinates (x, y). <X coordinate> can be an integer from −480 to 480 and <Y coordinate> can be an integer from −999 to 999.
The RMOVE statement is only valid in the graphics mode.

Example

```
10 PMODE GR
20 PMOVE 240, 0
30 PLINE 240, 0, 360, 120
40 RMOVE −120, 0
50 PLINE 240, 120, 360, 240
60 PMODE TN
```

The above program draws two oblique lines.
Remember to advance the paper before executing this program.

PHOME

Format **PHOME**

Abbreviated Format

PH.

Explanation The PHOME statement returns the pen to the origin. This statement is valid only in the graphics mode.

Example

```
10 PMODE GR
20 PLINE 240, −240
30 PCIRCLE 240, −240, 50
40 PHOME ..... Returns the pen to the home position.
50 PMODE TN
```

HSET

Format **HSET**

Abbreviated Format

H.

Explanation The HSET statement sets the current pen location as the new origin. The most appropriate location for drawing figures can be set as the origin by moving the pen to the location with a PMOVE statement before specifying a HSET statement. This statement is only vaid in the graphics mode.

Example

```
10 PMODE GR
20 PMOVE 240,-240
30 HSET ..... Sets (240,-240) as the new origin (0,0).
40 PMOVE 240,0
50 PLINE 240,0,0,-240,-240,0,0,240,240,0
60 PHOME
70 PMODE TN
```

GPRINT

Format **GPRINT** [[<size>,<angle>],] <text data>

Abbreviated Format

GP.

Explanation The GPRINT statement prints the specified character using the specified size and angle.

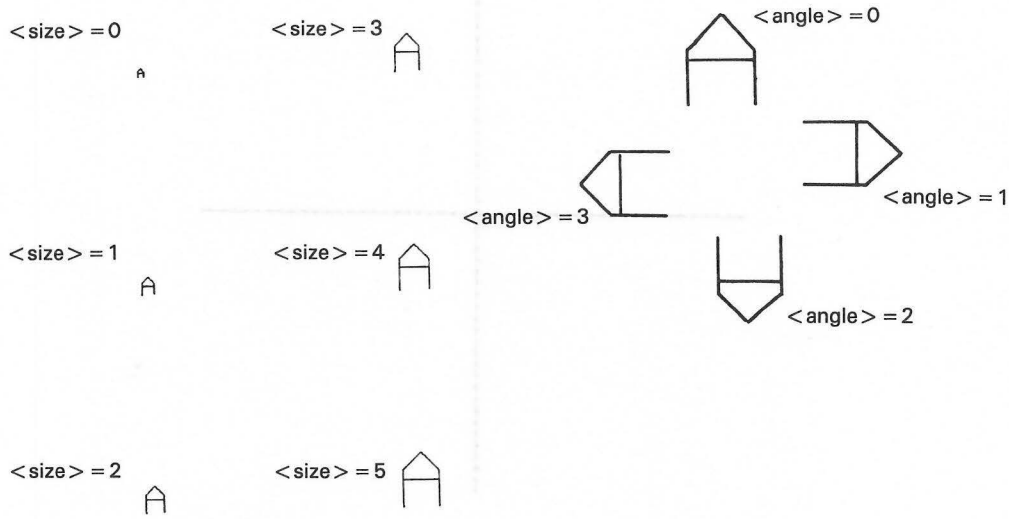
<size> may be any number from 0 to 63. 80 characters can be printed per line when <size> = 0; 40 characters per line when <size> = 1; and 26 characters per line when <size> = 2. <angle> indicates the direction in which character lines are printed. The character is rotated with respect to its lower left corner by the angle specified with <angle>. <angle> must be an integer from 0 to 3. When <size> and <angle> are omitted, the previous or default settings are assumed. The initial (default) values are <size> = 1 and <angle> = 0.

The GPRINT statement is only valid in the graphics mode.

Example

```
10 PMODE GR
20 GPRINT "A" ..... Prints "A" in the graphics mode.
30 PMOVE 240,-240
40 GPRINT [2,2],"A" ..... Prints an upside down "A" in the 26 characters/line
50 PHOME                  mode.
60 PMODE TN
```

The following figures show various examples of printout.



AXIS

Format **AXIS** <axis>,<pitch>,<repetitions>

Abbreviated Format

AX.

Explanation

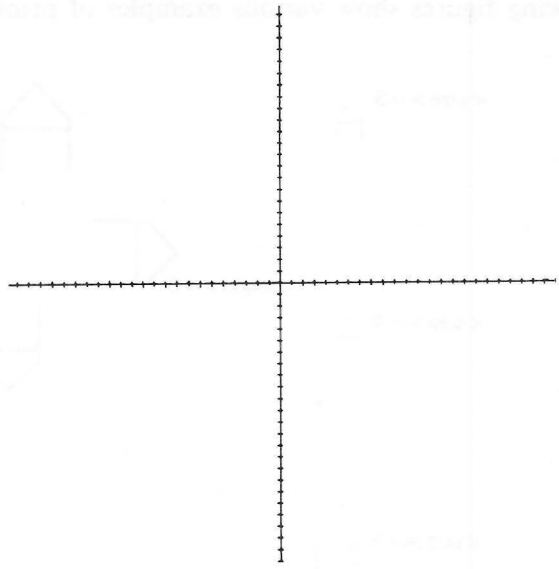
The AXIS statement draws the X-axis when <axis> = 1, and the Y-axis when <axis> = 0. The number of scale marks specified in <repetitions> is drawn with the pitch specified in <pitch>.

<pitch> must be an integer from −999 to 999. <repetitions> must be an integer from 1 to 255.

The AXIS statement is only valid in the graphics mode.

Example

```
10 PMODE GR
20 PMOVE 240, 0
30 AXIS 0,-10,48
40 PMOVE 0,-240
50 AXIS 1,10,48
60 PMODE TN
```



The above example draws the X and Y axes with scale marks from −240 to 240 at 10 unit intervals.
The coordinates can be used in the same manner as ordinary Cartesian coordinates after setting the point of intersection of the X and Y axes as the new origin. (X = −240 to 240, Y = −240 to 240)

PCIRCLE

Format

PCIRCLE <X coordinate>, <Y coordinate>, <radius>,
<starting angle>, <ending angle>, <step angle>
<X coordinate>: −999 to 999
<Y coordinate>: −999 to 999
<radius>: 0 to 999

Abbreviated Format

PCI.

Explanation

The PCIRCLE statement draws a circle, or arc counterclockwise. The circle (arc) has a <radius> and a <step angle>, with the center at location (x,y), and starts at <starting angle> and ends at <ending angle>. A complete circle is drawn when <starting angle> = 0, <ending angle> = 360, and <step angle> = 0.2.
This statement actually draws a polygon, therefore <step angle> must be as small as possible in order to draw a smooth figure. <starting angle> must be smaller than <ending angle>. When <step angle> = 0, lines connecting the center and the starting point and the center and the ending point are drawn. The PCIRCLE statement is only valid in the graphics mode.

Example

```
10 PMODE GR:P=0
20 PMOVE 240,-240
30 HSET
40 FOR J=240 TO 40 STEP -60
50 PCOLOR P
60 PCIRCLE 0, 0, J, 0, 360, 2
70 P=P+1
80 NEXT J
90 PMODE TN
```

LIST/P

Format **LIST/P** [<starting line number>] [−] [<ending line number>]

Abbreviated Format

L./P

Explanation

The LIST/P command lists all or part of the program lines in memory on the printer. See the explanation of the LIST command for an explanation of procedures for specifying the range of lines to be printed. Note that, when graphic characters are included in the program list, most of them will be printed in a different colour as hexadecimal ASCII codes if the plotter printer is used.
This statement is valid only in the text mode.

HCOPY

Format **HCOPY**

Abbreviated Format

HC.

Explanation

The HCOPY command copies the contents of the screen onto the printer. This command is only available for the MZ-80P5(K) printer and cannot be used for the colour plotter printer.

PLOT

Format **PLOT** { ON / OFF }

Abbreviated Format

PL.

Explanation

The PLOT ON statement makes it possible to use the colour plotter printer as a display unit. Thus, the MZ-800 can be used without an external display screen.
The PLOT ON statement sets the number of characters printed per line to 80 when the screen is in the 80-column mode and sets it to 40 when the screen is in the 40-column mode.
This statement is only valid when the colour plotter printer is installed and used in the text mode. The CONSOLE command is made invalid once a PLOT ON is executed.
A period "." is printed to represent any character which is not contained in the colour plotter printer's character generator. The INST , DEL , and ← keys are disabled by executing this statement. CTRL + G can be used to change the colour of the pen.
The PLOT OFF command cancels the PLOT ON command.
The INIT"CRT:Mn" statement also cancels the PLOT ON command.
The printer is set to the 40-character mode if the PLOT ON is executed when the display is in the 40-character mode; it is set to the 80-character mode if the statement is executed when the display is in the 80-character mode.

6.7 Machine Language Control Statements

PEEK

Format

PEEK <address>

Explanation

This function returns the contents of the specified address as a decimal number from 0 to 255. <address> may be a decimal number from 0 to 65535 or a 4-digit hexadecimal number from $0000 to $FFFF.

Example

The following program displays data stored in the area from 40960 ($A000) to 40975 ($A00F):

```
10 FOR AD=40960 TO 40975
20 PRINT PEEK (AD)
30 NEXT AD
40 END
```

POKE

Format

POKE <address>, <data>[, <data>] ...

Explanation

The POKE statement writes a consecutive number of data values starting at the specified address.
<address> may be a decimal number from 0 to 65535 or a 4-digit hexadecimal number from $0000 to $FFFF. <datan> may range from 0 to 255 or from $00 to $FF. This statement can write data to any memory location, regardless of the limit set by the LIMIT statement. Therefore, careless use of this statement can destroy the monitor or BASIC interpreter.

Example

```
POKE $D000,$5F ..... Uses hexadecimal numbers.
POKE 53248,95 ..... Uses decimal numbers.
```

The two statements above perform the same function.

INP@

Format

INP@ <port number>, <variable>

Explanation

The INP@ statement reads 8-bit data from the input port specified in <port number>, converts it into a decimal number and assigns it to <variable>. <port number> may be in the range 0 to 127 (hexadecimal $00 to $7F). Port addresses 128 to 255 ($80 to $FF) are reserved for optional peripheral devices.

OUT@

Format

OUT@ <port number>, <numeric expression>

Explanation

The OUT@ statement converts the decimal number specified in <numeric expression> (0 to 255) to a binary format and sends it to the output port specified in <port number>. <port number> may range from 0 to 127 (hexadecimal $00 to $7F). Port addresses 128 to 255 (hexadecimal $80 to $FF) are reserved for optional peripheral devices.

Peripheral devices are controlled by data transmitted to I/O ports. Consequently, specifying an illegal number in <port number> may cause peripheral device malfunction.

USR

Format

USR (<address>[,<input string variable>] [,<output string variable>])

Abbreviated Format

U.

Explanation

The USR function transfers control to a machine language program which starts at the specified address. As with CALL <address>, control is returned to the statement following the USR function when a return instruction RET or RET cc is encountered in the machine language program. <address> must be a decimal or 4-digit hexadecimal number.

The parameters are loaded into the following registers when the main program transfers control to the machine language program:

DE register: Starting address of <input string variable> in memory.

B register: length of <input string variable>.

IX register: address of the error processing routine, if declared.

The machine program loads processing results into the following registers when it returns control to the main program:

DE register: starting address of <output string variable> in memory.

B register: length of <output string variable>.

The following steps are necessary when error processing is required in the machine language program:

1. Declare an error processing routine in the BASIC program using an ON ERROR GOTO statement.
2. Write a program segment which loads the A register with the error code and causes program execution to jump to the address specified in the IX register.

LIMIT

Format

LIMIT <address>
LIMIT MAX

Abbreviated Format

LIM.

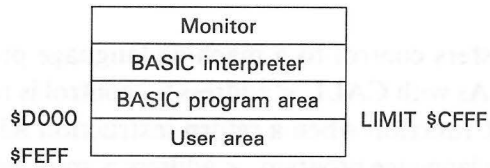
Explanation

The LIMIT statement limits the memory space available for use by BASIC. <address> sets the upper limit of the BASIC area; the area following that address to $FEFF (65279) are set aside as the user area. The area from $FF00 to $FFFF is used by the monitor as a work area, so it cannot be used for user programs. <address> can either be a decimal number or 4-digit hexadecimal number. When linking a BASIC program with a machine language program or storing special data in memory, sufficient memory space must be reserved for the user area.
The LIMIT statement must appear at the beginning of the program. The LIMIT MAX statement releases the limit specified by a LIMIT statement.

Example

```
LIMIT $CFFF
```

Limits the BASIC program area to $CFFF and defines the area above that address as the user area.




```
LIMIT MAX
```

Resets the limit established by a previous LIMIT statement.

6.8 Error Processing Statements

ON ERROR GOTO

Format

ON ERROR GOTO { <line number> / <label> }

Abbreviated Format

ON ERR. G.

Explanation

The ON ERROR GOTO statement causes program execution to branch to <line number> or <label> if an error occurs. The ERN or ERL system variable can be used in a trap routine starting at that line to control subsequent processing according to the type of error and the line number in which it occurred. Including a RESUME statement at the end of the error processing routine makes it possible to return execution to the line at which the error occurred. Executing an ON ERROR GOTO statement cancels the error trap line number defined by the previous ON ERROR GOTO statement. The error trap line number definition is also cancelled by executing a CLR statement.

Example

```
10 ON ERROR GOTO 100
20 INPUT "X=";X
30 PRINT SQR(X)
40 END
100 PRINT "ERROR"
110 RESUME 20
```

The program above displays the message "ERROR" and returns to line 20 if an error occurs.

RESUME

Format

RESUME <line number>
RESUME NEXT
RESUME 0
RESUME

Abbreviated Format

RESU.

Explanation

The RESUME statement returns control to the main routine from an error processing routine.
The system holds the number of the line on which the error occurred in memory and returns program execution to that line or to another specified line after the error is corrected. The RESUME statement may be used in any of the following four forms:

RESUME ..... returns to the error line.
RESUME NEXT ..... returns to the line following the error line.
RESUME <line number> ..... returns to the line specified in <line number>.
RESUME 0 ..... returns to the beginning of the main routine.

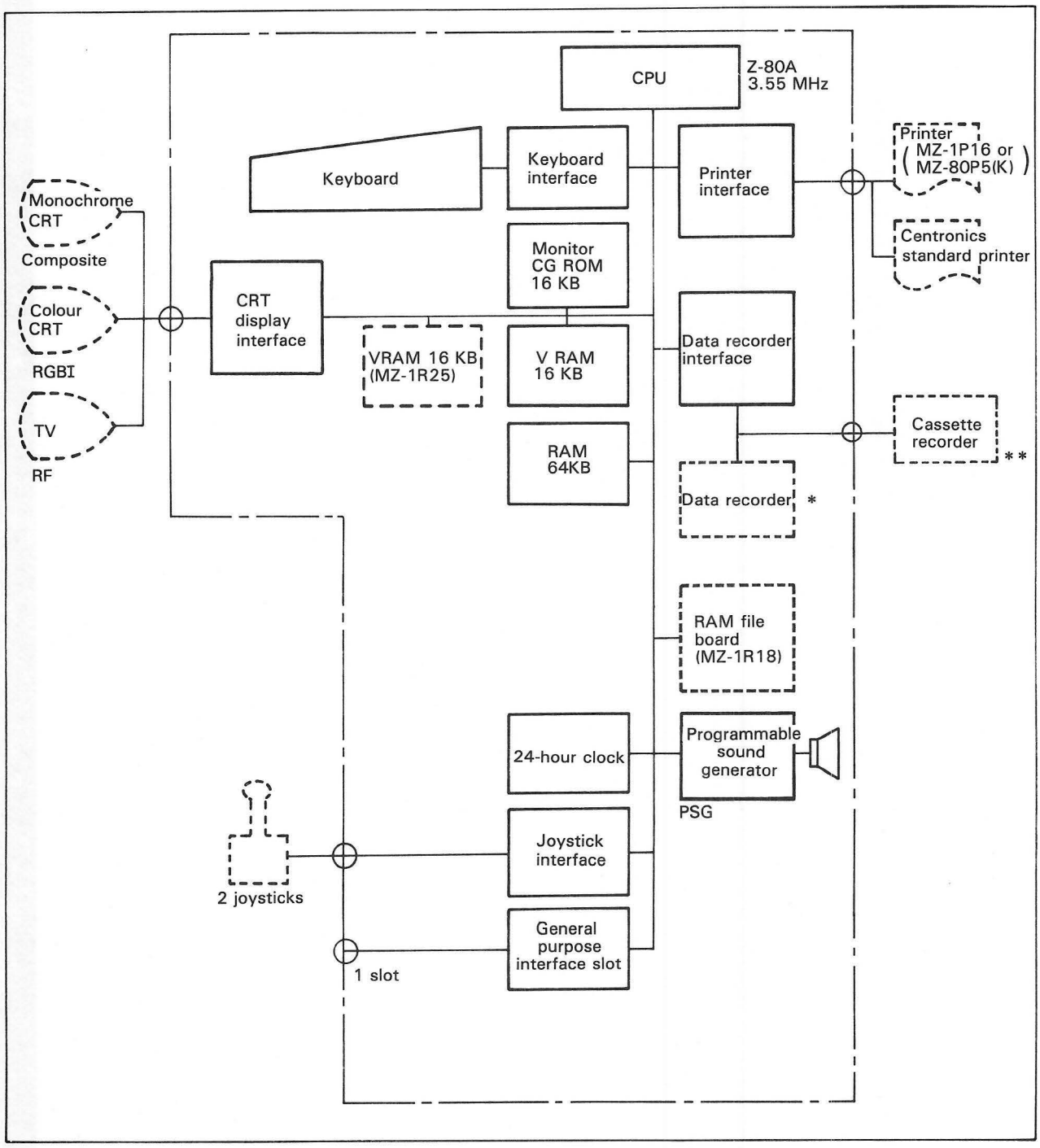
Always use a RESUME statement to return to the main program from the error processing routine.
If RESUME is encountered when no error has occurred, an error occurs.
If RESUME cannot be executed, an error occurs.

Chapter 7 Hardware

This chapter describes the MZ-800 hardware. It also describes peripheral devices which can be connected to the MZ-800 and how to connect them.

7.1 MZ-800 Hardware

7.1.1 System diagram



Optional device

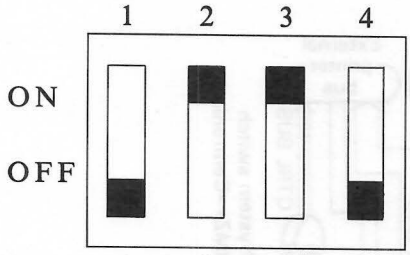
* Standard with the MZ-821 and optional with the MZ-811.

** Can only be used with the MZ-811.

7.1.2 System switch settings

A 4-switch DIP switch package is located at the rear of the MZ-800. These switches are called the system switch. The function of each switch is as follows.

Note:
Be sure to turn off the power switch when setting the system switch.



	ON	OFF
Not used	—	—
Printer	MZ	Centronics
Operational mode	MZ-700	MZ-800

Switch 1: Mode switch
This switch is used to switch the operating mode between the MZ-700 and MZ-800 modes. Normally, this switch is OFF. (See Chapter 9 for the MZ-700 mode.)

ON: MZ-700 mode
OFF: MZ-800 mode

Switches 2 and 3: Printer interface selection
These switches are used to switch the interface between the MZ printer system and Centronics system. Both switches must be set to the same position.

ON: MZ printer
OFF: Centronics interface

If your Centronics standard printer does not operate even if both switches are set to OFF, set either switch to ON.

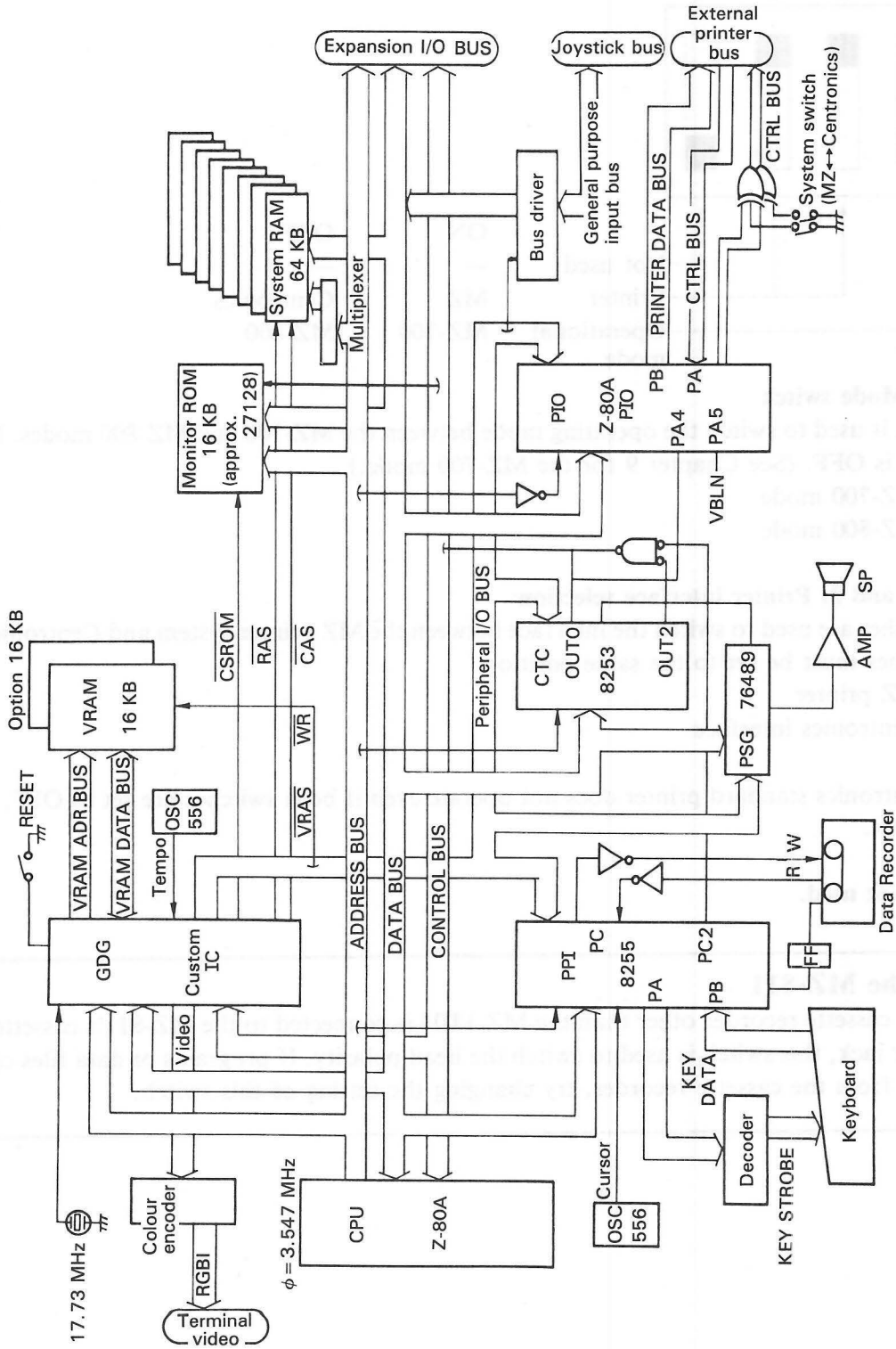
Switch 4: Not used.

> **With the MZ-811**
> When a cassette recorder other than the MZ-1T04 is connected to the MZ-811's cassette tape recorder jack, this switch is used to switch the head polarity. If programs or data files cannot be read from the cassette recorder, try changing the setting of this switch.

7.1.3 I/O port control

The following information is provided for reference only. Caution must be exercised when I/O ports are utilized by the user, as it is possible to create machine malfunctions or to disable peripheral operation if I/O port operation is incorrect.

(1) Block diagram



(2) I/O port address

The following I/O port addresses are already assigned to the existing I/O devices or are reserved for peripheral devices which Sharp has planned for the future.

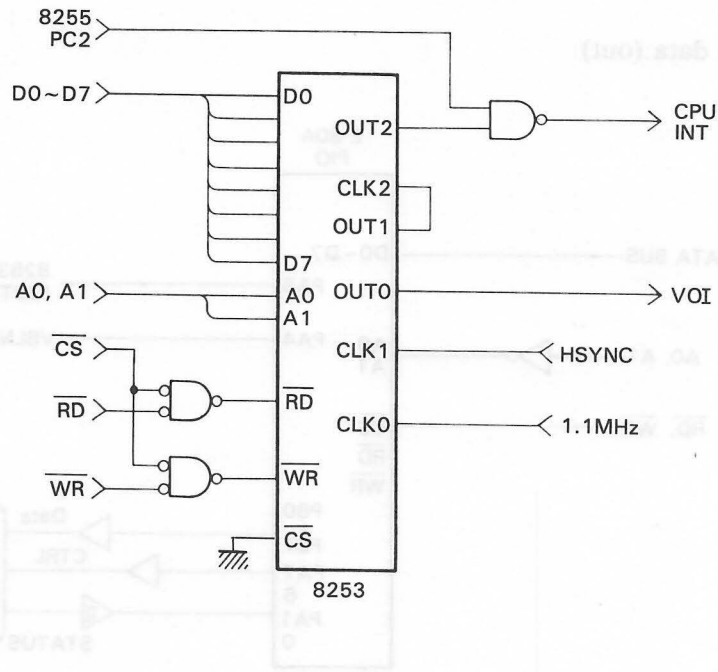
B0 to B3 : Serial I/O port
CC to CF : GDG (graphic display generator)
D0 to D3 : 8255 (data recorder and keyboard control)
D4 to D7 : 8253 (programmable interval timer)
D8 to DF : FDC (floppy disk controller)
E0 to E6 : GDG
F0 to F1 : Joystick inputs
F2 : PSG output
F4 to F7 : QDC
FC to FF : Z-80A PIO (printer)

(3) Programmable clock generator (8253)

The MZ-800 has a built-in programmable interval timer. This timer is used for controlling the built-in clock and programmable sound generator.
In the MZ-700 mode, memory mapped I/O addresses $E004 to $E007 are assigned to this timer, while in the MZ-800 mode, I/O mapped addresses $D4 to $D7 are assigned.

$D4 : counter 0
$D5 : counter 1
$D6 : counter 2
$D7 : control word register

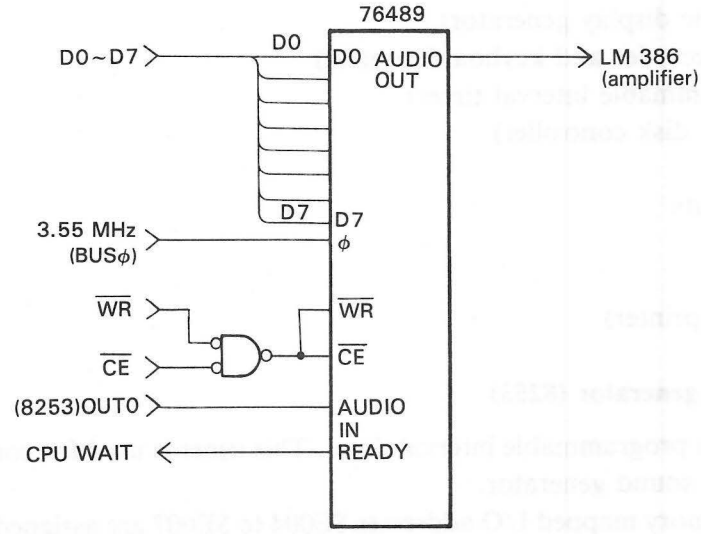
Counter 0 is used for the programmable sound generator, counter 1 is used internally and counter 2 is used for interrupting the CPU.



(4) Programmable sound generator (76489)

The MZ-800 has a built-in programmable sound generator (PSG) which can generate 3-tone chords over 8 octaves.

I/O port address $F2 is assigned to the PSG. For details on controlling the PSG, see Appendix B.



(5) Printer interface (Z-80A PIO)

The MZ-800 uses a Z-80A PIO for the printer interface.

I/O port addresses FC to $FF are assigned to the PIO.

$FC: Control register

$FD: Control register

$FE: Port A

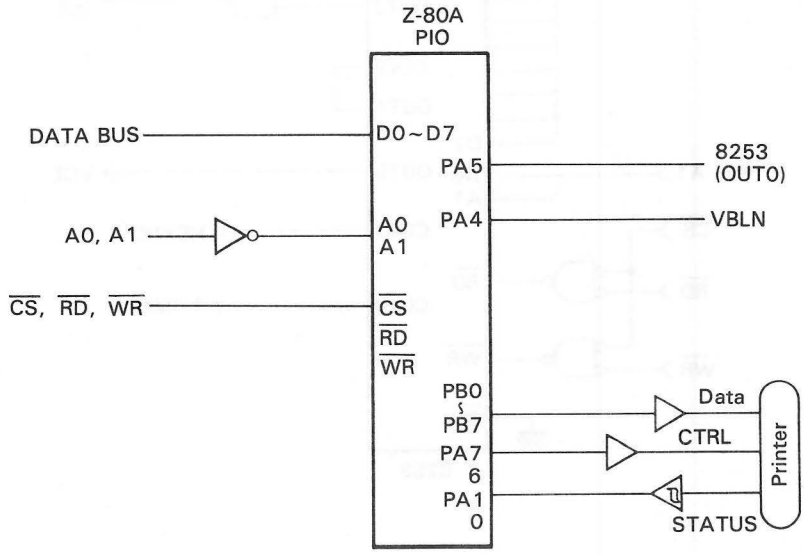
PA0, PA1: printer status (in)

PA4, PA5: system status (in)

PA6, PA7: printer control (out)

$FF: Port B

Printer data (out)



(6) Keyboard and data recorder controller (8255)

The MZ-800 uses an 8255 to control the keyboard and data recorder.

In the MZ-700 mode, memory mapped I/O addresses $E000 to $E003 are assigned to 8255 while in the MZ-800 mode, I/O mapped addresses $D0 to $D3 are assigned.

$D0: Port A

PA0 to PA3: KEYSTROBE signals (out)
PA4 to PA5: JOYSTROBE signals (out)
PA7: cursor RST (out)

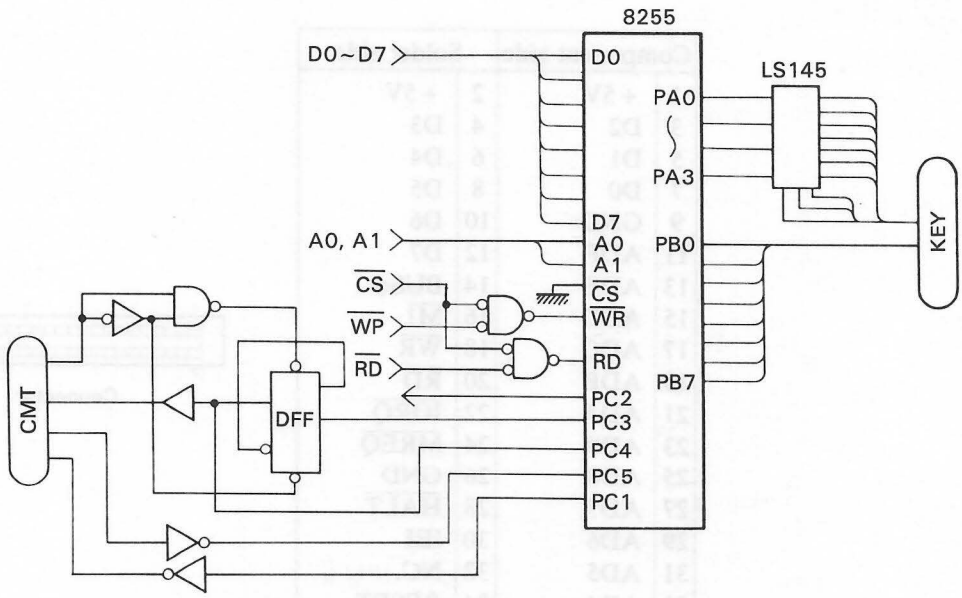
$D1: Port B

PB0 to PB7: KEYDATA signals (in)

$D2: Port C

PC0: SOUND MASK (out)
PC1: CMTWR (out)
PC2: disable INT (out)
PC3: MOTOR (out)
PC4: SENSE (in)
PC5: CMTRD (in)
PC6: cursor FLSH (in)
PC7: VBLNK (in)

$D3: Control register



7.2 Peripheral Devices

Many optional peripheral devices are available, but some of those explained in this manual may not be available in your country.
Be sure to turn off the power switches of both the MZ-800 and peripheral device when connecting them.

7.2.1 Standard interfaces

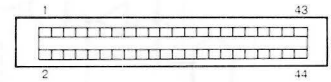
The MZ-800 is equipped with the following interfaces as standard.

- CRT display interface
- Keyboard interface
- Data recorder interface
- Printer interface
- Joystick interface

7.2.2 Expansion I/O connector

An expansion I/O connector is provided inside the computer, which can be accessed by removing the expansion slot cover from the rear panel. This connector is provided for the connection of an optional interface or expansion unit. The pin assignment of the expansion I/O connector is as follows.

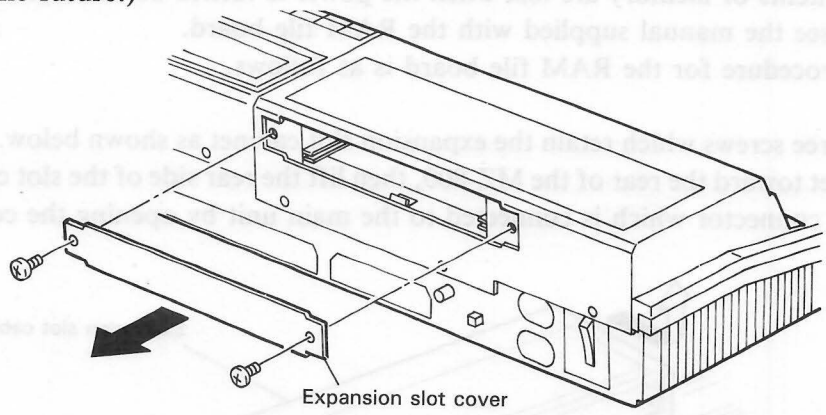
Component side		Solder side	
1	+5V	2	+5V
3	D2	4	D3
5	D1	6	D4
7	D0	8	D5
9	GND	10	D6
11	ADF	12	D7
13	ADE	14	BUSφ
15	ADD	16	$\overline{M1}$
17	ADC	18	$\overline{WR}$
19	ADB	20	$\overline{RD}$
21	ADA	22	$\overline{IORQ}$
23	AD9	24	$\overline{MREQ}$
25	AD8	26	GND
27	AD7	28	$\overline{HALT}$
29	AD6	30	IEI
31	AD5	32	NC. INH 5
33	AD4	34	RESET
35	AD3	36	$\overline{EXRESET}$
37	AD2	38	$\overline{INT}$
39	AD1	40	$\overline{EXWAIT}$
41	AD0	42	NC. SOUND
43	GND	44	GND
45	GREEN	46	YITN
47	BLUE	48	H-SYN
49	RED	50	V-SYN



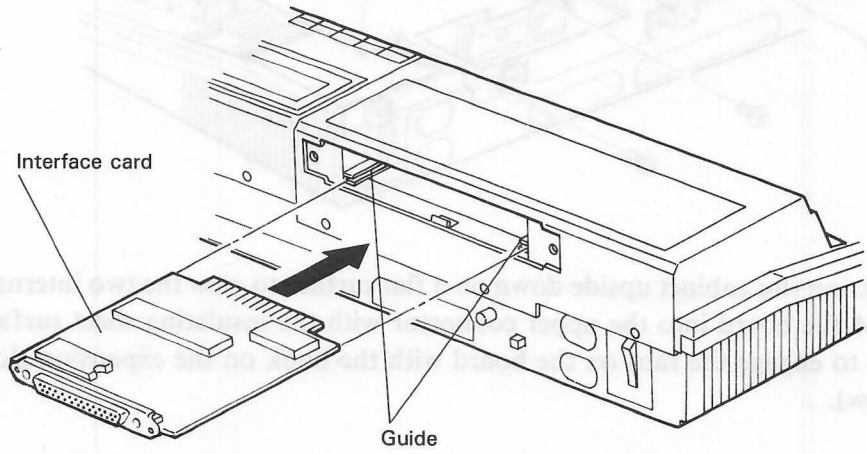
Connector

• Installation of an optional interface

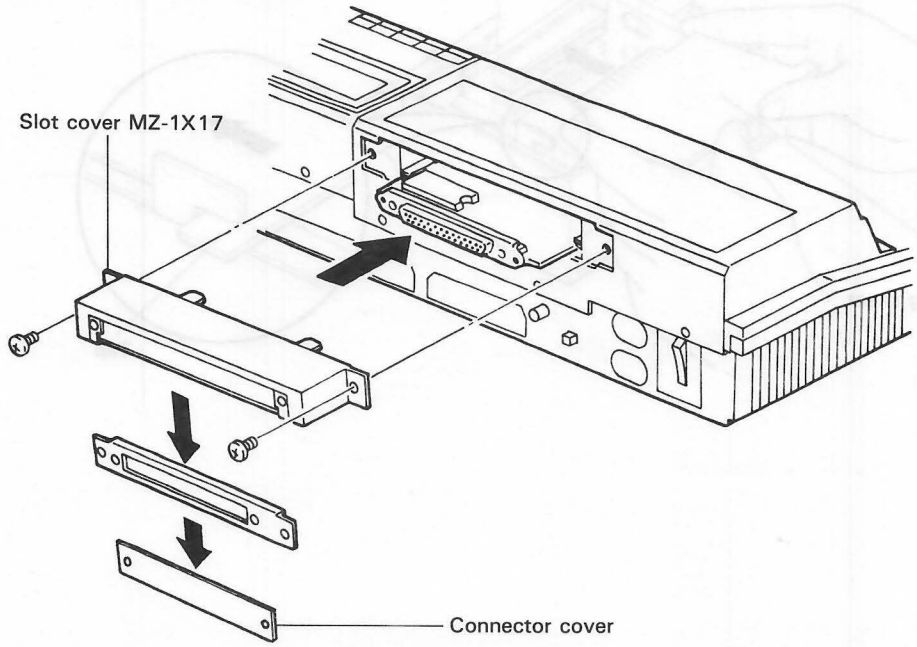
1) Remove the expansion slot cover. (Store the cover in a safe place in case you want to remove the interface in the future.)



2) Insert the interface card into the slot, and slide it along the card guides with the component side up. Firmly press the card into the expansion I/O connector at the rear of the slot.



3) Remove the connector cover from optional slot cover MZ-1X17. Affix the optional slot cover to the rear panel of the MZ-800.

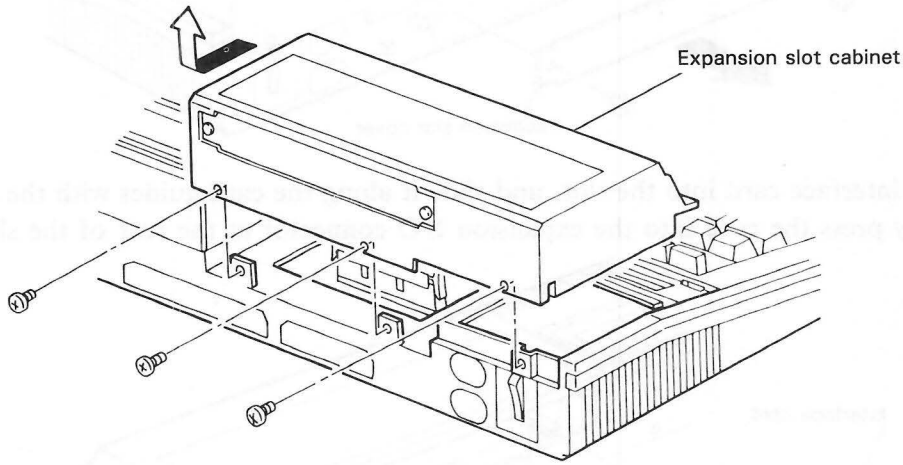


For details, see the manual supplied with the interface.

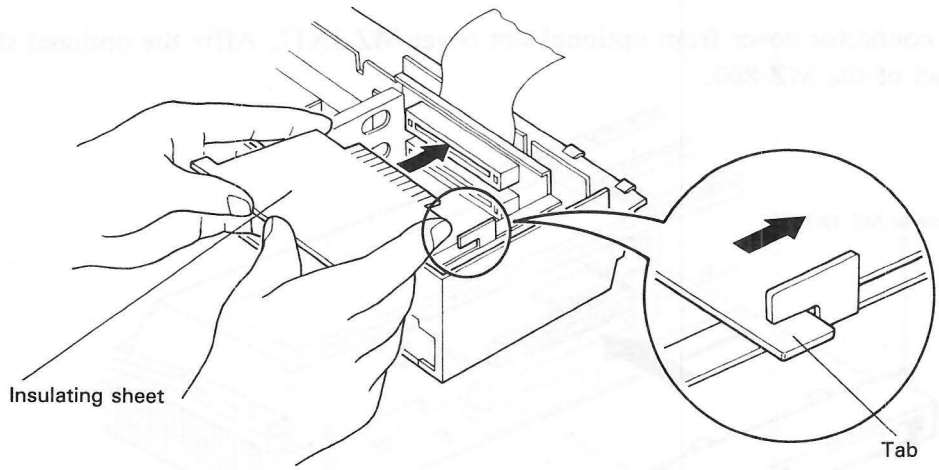
7.2.3 RAM file board (MZ-1R18)

The RAM file board is a memory device which can be used in the same manner as floppy disk drives, except that the contents of memory are lost when the power is turned off. For full details about the RAM file board, see the manual supplied with the RAM file board.
The installation procedure for the RAM file board is as follows.

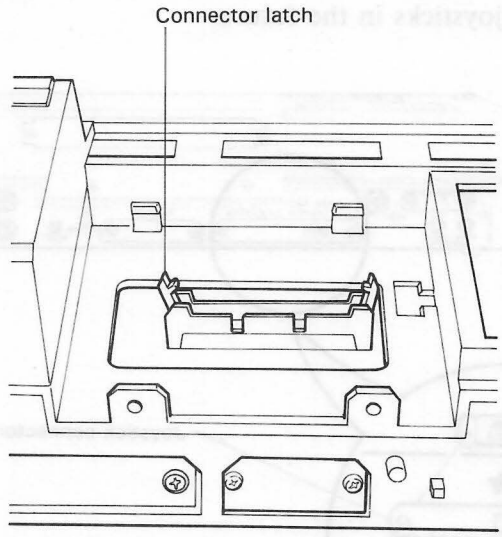
(1) Remove the three screws which retain the expansion slot cabinet as shown below. Push the expansion slot cabinet toward the rear of the MZ-800, then lift the rear side of the slot cabinet to remove it. Unplug the connector which is connected to the main unit by opening the connector latches.



(2) Place the expansion slot cabinet upside down on a flat surface to view the two internal connectors. Insert the RAM file board into the upper connector with the insulating sheet surface facing upwards. Be sure to engage the tabs on the board with the hook on the expansion slot chassis (see the figure below).



(3) Plug the connector previously unplugged in Step 1 into the connector on the main unit. Replace the expansion slot cabinet and secure it with the three screws. Close the connector latches to firmly hold the connector.



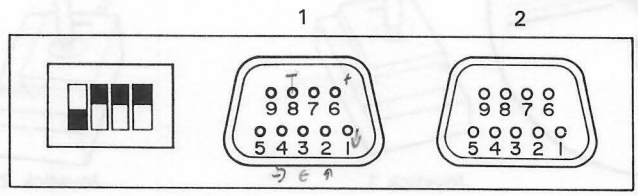
Note:
If the cable is trapped between the cabinet and main unit, the expansion slot cabinet cannot be replaced properly.

7.2.4 Joystick

Joysticks made by Atari Inc., or their equivalents can be used with the MZ-800.

JOYSTICK 1	
1	FWDA
2	BACKA
3	LEFTA
4	RIGHTA
5	+5V
6	TRG1A
7	TRG2A
8	COMA
9	GND

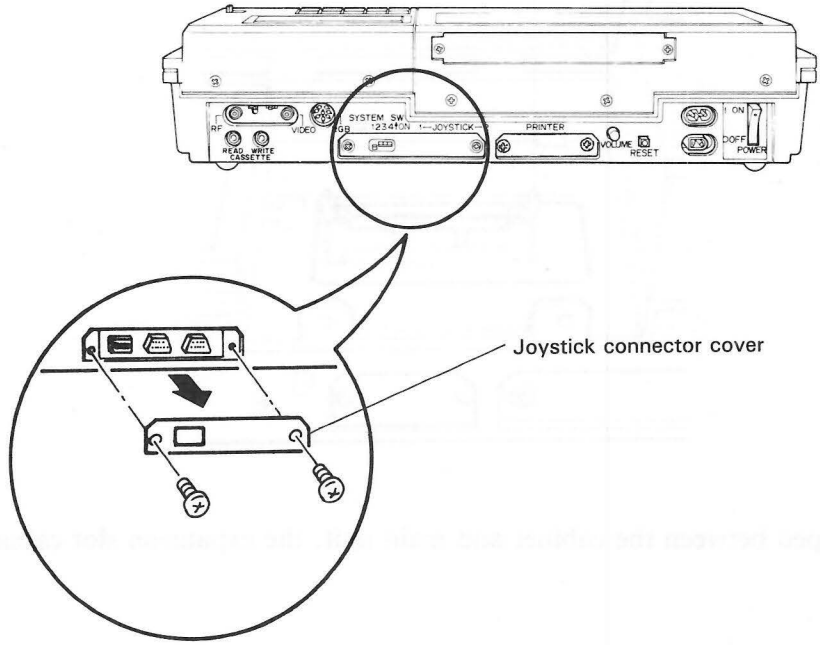
JOYSTICK 2	
1	FWDB
2	BACKB
3	LEFTB
4	RIGHTB
5	+5V
6	TRG1B
7	TRG2B
8	COMB
9	GND



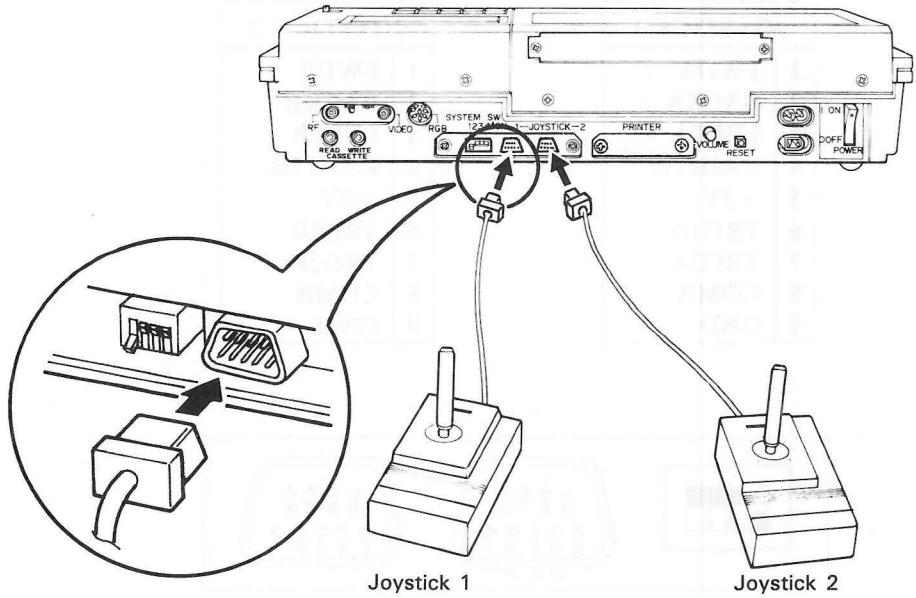
Connector

The connection procedure is as follows.

1) Remove the joystick connector cover from the rear panel. Store the cover in a safe place in case you want to disconnect the joysticks in the future.



2) Plug in the cables from the joysticks as shown below.



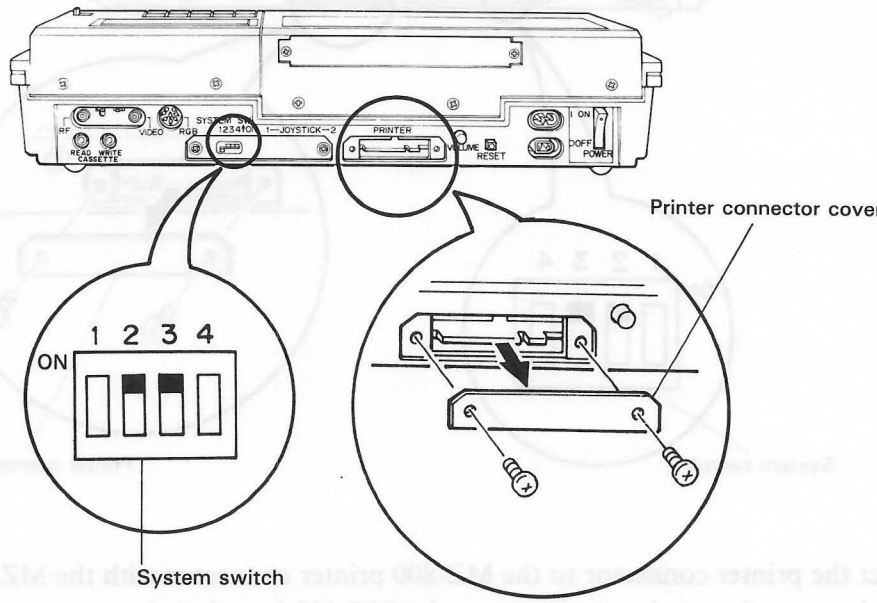
7.2.5 Printers

Various types of printers can be used with the MZ-800, including two SHARP printers.

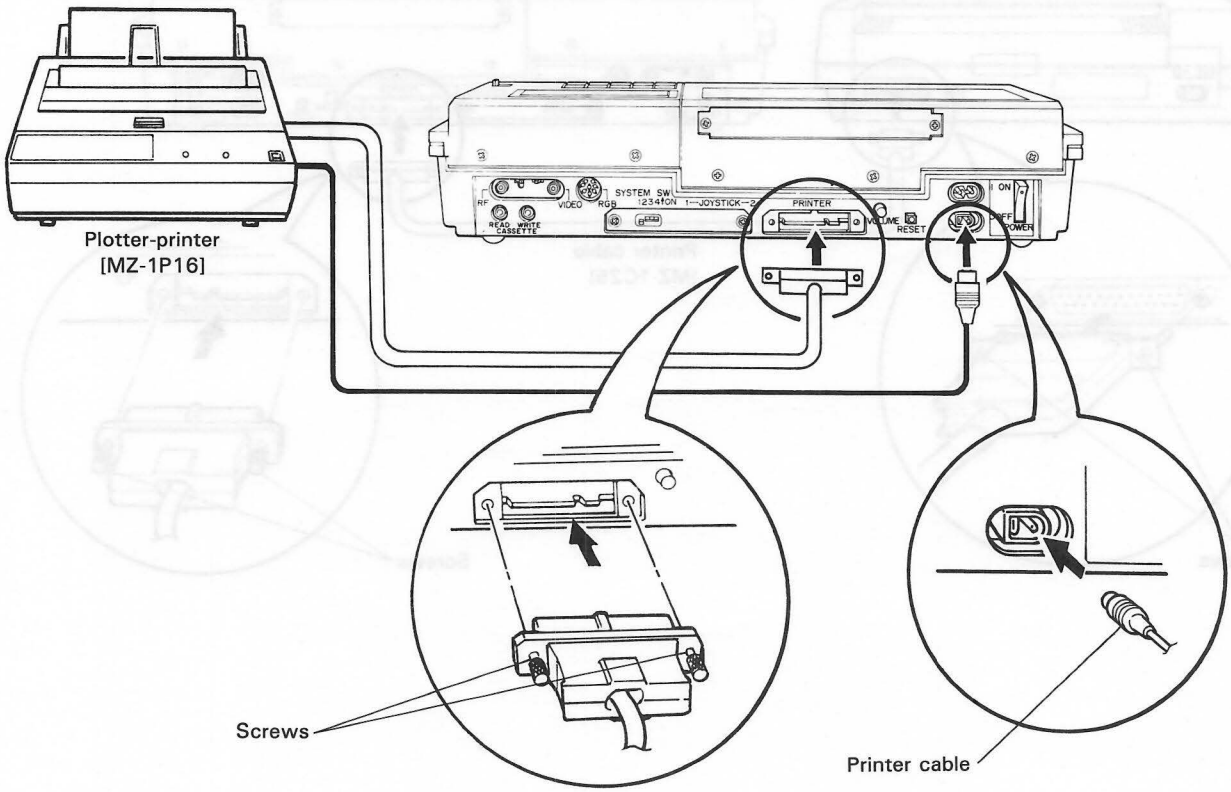
(1) Plotter-printer MZ-1P16

Connecton procedure is as follows.

1) Set switches 2 and 3 of the system switch to the ON position.
2) Remove the two screws to remove the printer connector cover from the rear panel of the MZ-800. (Store the cover in a safe place in case you want to disconnect the printer cable in the future.)



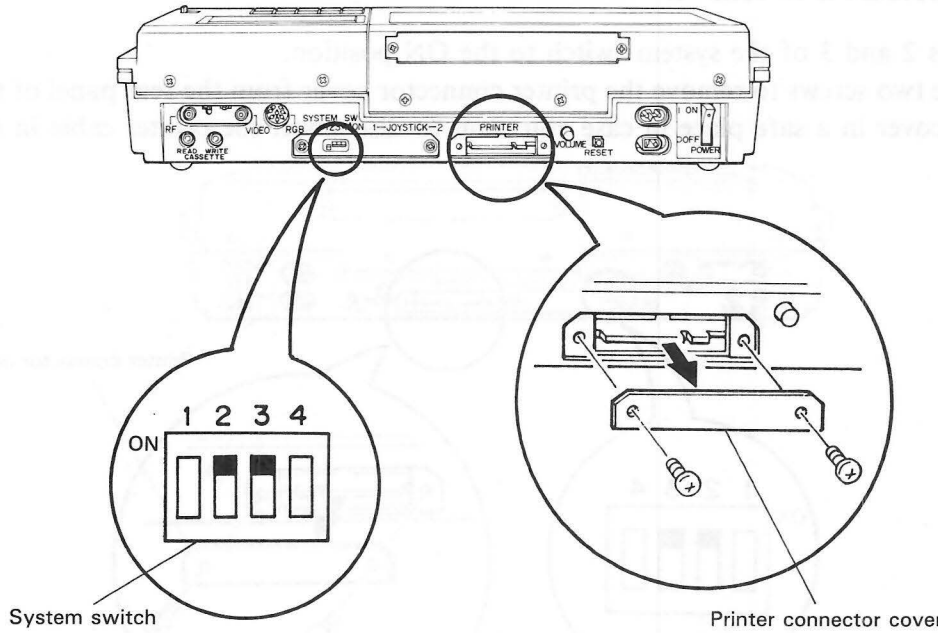
3) Plug the printer cable connector into the MZ-800 card edge connector, with the connector key facing upwards, and fasten the connector to the MZ-800 with the screws.



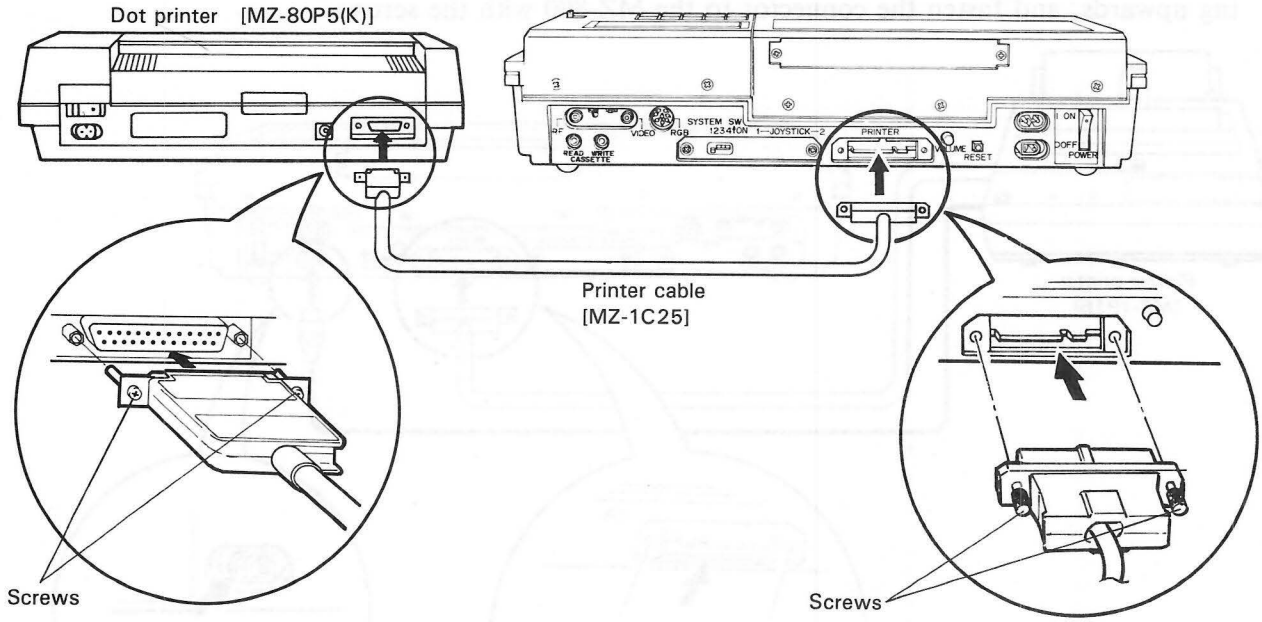
4) Plug the printer power cable into the plotter power jack on the rear panel of the MZ-800.

(2) Dot matrix printer MZ-80P5(K)

1) Set switches 2 and 3 of the system switch to the ON position.
2) Remove the two screws to remove the printer connector cover from the rear panel of the MZ-800. (Store the cover in a safe place in case you want to disconnect the printer cable in the future.)

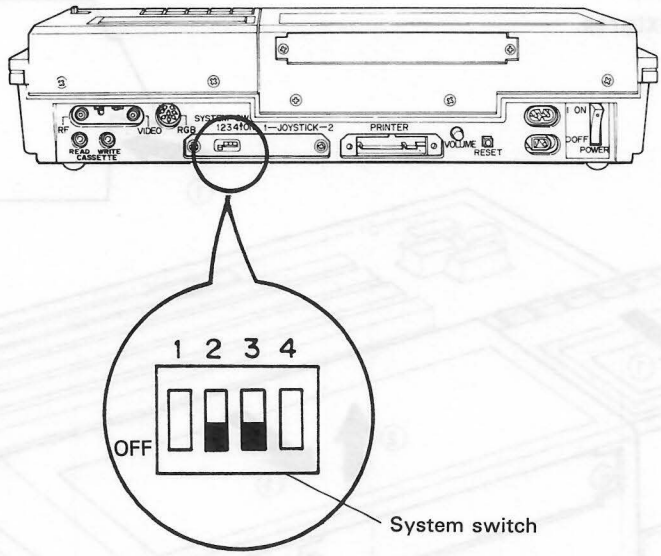


3) Connect the printer connector to the MZ-800 printer connector with the MZ-1C25 optional cable. Remember to refasten the connector to the MZ-800 by using the screws.



(3) Other printers

Switching switches 2 and 3 of the system switch OFF allows you to use a printer equipped with a Centronics interface. However, some commercially available printers which are sold as Centronics standard printers do not actually comply with the Centronics Standard and therefore cannot be used. Some printers have character code sets different from that used by the MZ-1P16 or MZ-80P5(K) printer. These types of printers can be used but may require special programming to allow full utilization of all the features of the MZ-800.



Printer interface

Component side		Solder side	
1	RDP	2	GND
3	RD1	4	GND
5	RD2	6	GND
7	RD3	8	GND
9	RD4	10	GND
11	RD5	12	GND
13	RD6	14	GND
15	RD7	16	GND
17	RD8	18	GND
19	IRT	20	GND
21	$\overline{\text{RDA}}$	22	GND
23	$\overline{\text{STA}}$	24	GND
25	GND	26	GND

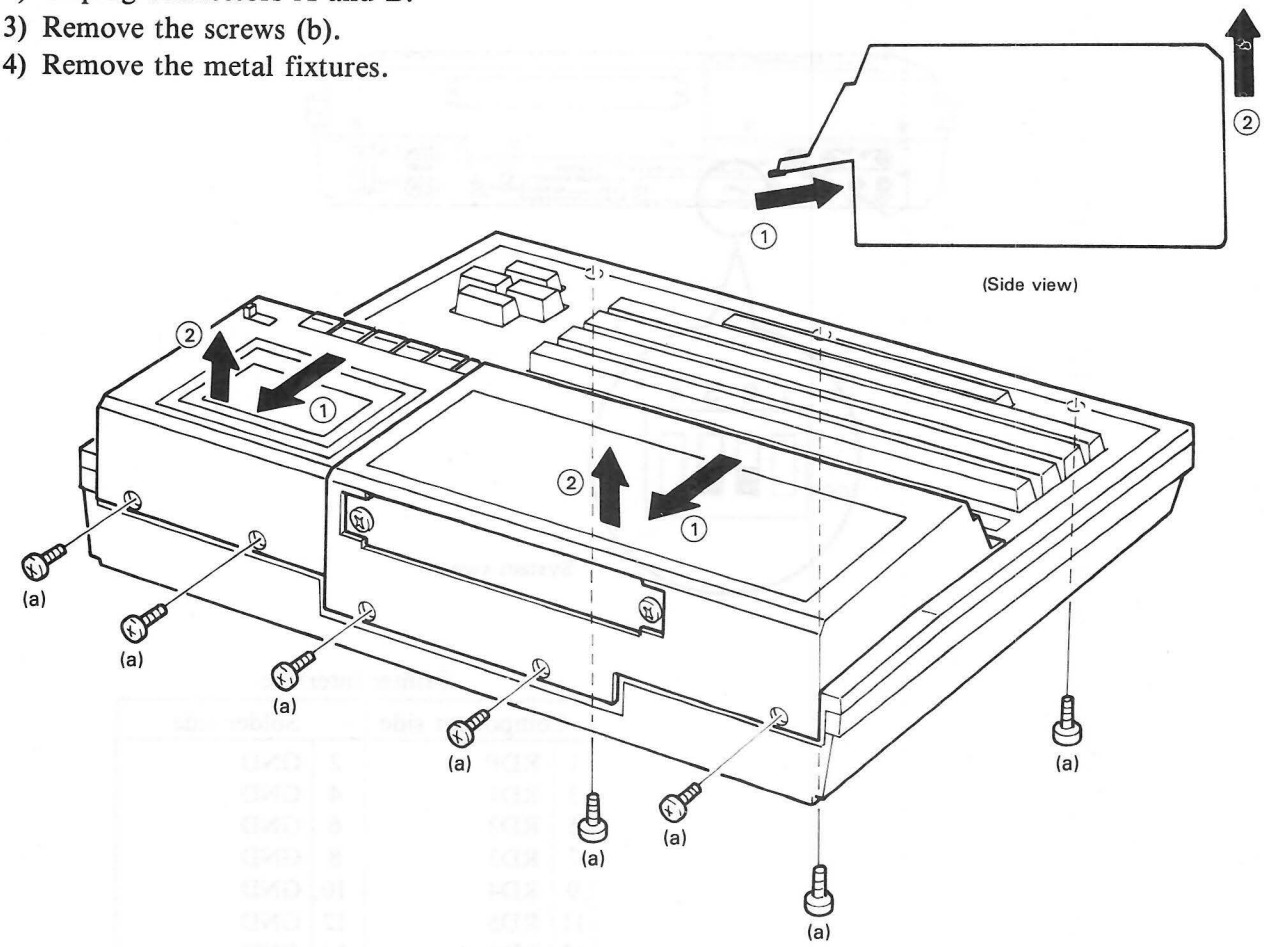


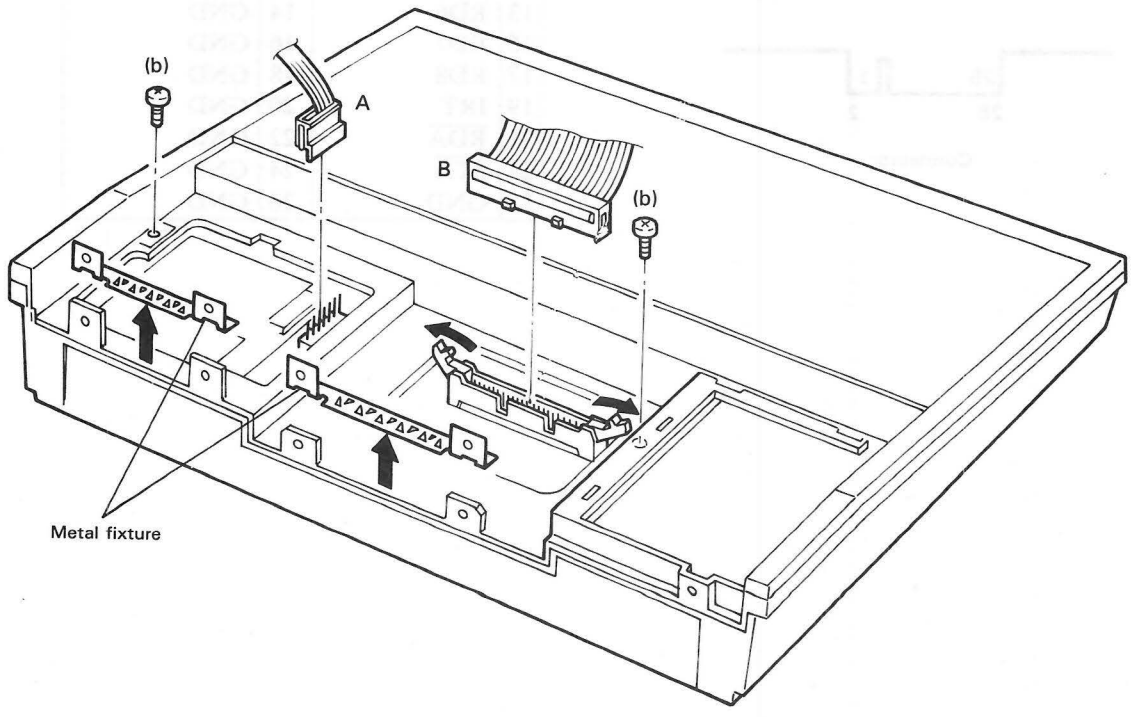
Connector

7.2.6 Optional graphic memory MZ-1R25

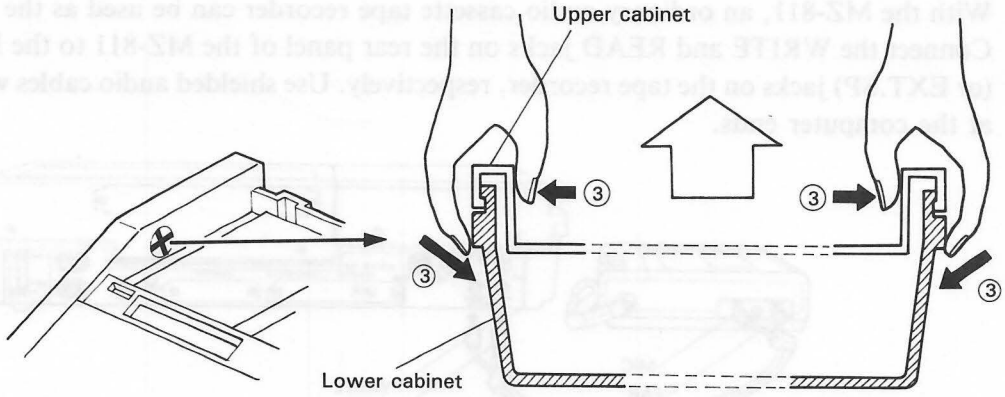
An optional MZ-1R25 graphic memory further improves the display capability of the MZ-800. The set includes two ICs which must be installed inside the cabinet. Follow the installation procedure below:

1) Remove screws (a) and detach the data recorder unit and expansion slot compartment cabinet as indicated by arrows ① and ② .
2) Unplug connectors A and B.
3) Remove the screws (b).
4) Remove the metal fixtures.

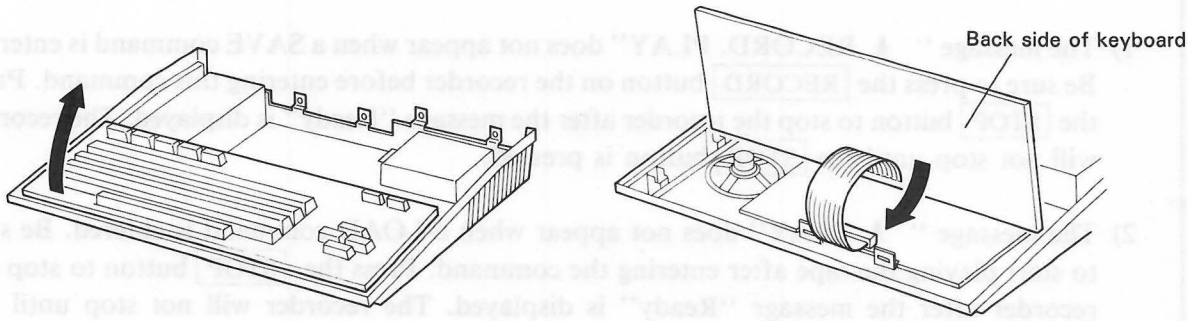




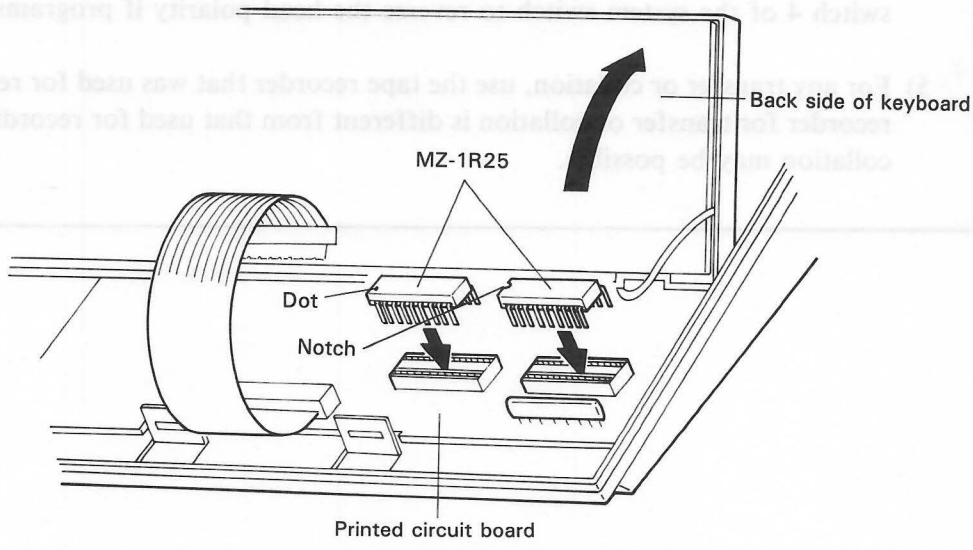
5) Press and hold the upper cabinet at the points indicated by the arrows ③, then pull up the upper cabinet to remove it from the lower cabinet.



6) Lift the front of the keyboard as shown below.



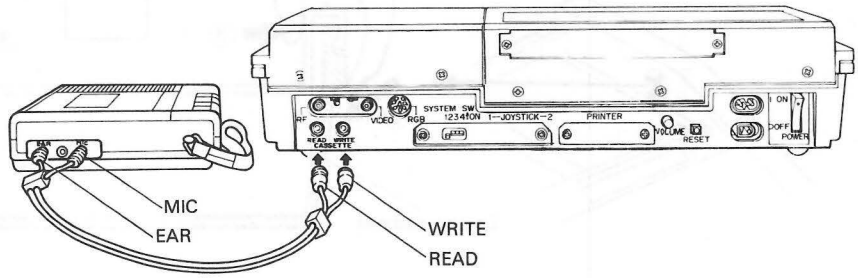
7) Two IC sockets are located near the front right corner of the main printed circuit board. Insert the IC chips into the IC sockets as shown below. Take care that you install the pointing chips in the correct direction (with the dot or notch over the first pin of the IC facing the center of the computer). Installing the chips in the wrong direction may damage them.



8) Perform steps 1) to 6) in the reverse order to reassemble the MZ-800.

7.2.7 External cassette tape recorder (for MZ-811 only)

With the MZ-811, an ordinary audio cassette tape recorder can be used as the data recorder. Connect the WRITE and READ jacks on the rear panel of the MZ-811 to the MIC and EAR (or EXT.SP) jacks on the tape recorder, respectively. Use shielded audio cables with 3.5 ø jacks at the computer ends.



Note the following when using an **ordinary cassette tape recorder.**

1) The message " ↓ RECORD. PLAY" does not appear when a SAVE command is entered. Be sure to press the RECORD button on the recorder before entering this command. Press the STOP button to stop the recorder after the message "Ready" is displayed. The recorder will not stop until the STOP button is pressed.

2) The message " ↓ PLAY" does not appear when a LOAD command is entered. Be sure to start playing the tape after entering the command. Press the STOP button to stop the recorder after the message "Ready" is displayed. The recorder will not stop until the STOP button is pressed.

3) The level and tone controls of the cassette tape recorder must be adjusted to appropriate levels. Some cassette recorders (e.g., those with an automatic level control) may not be usable. In such cases, please purchase the MZ-1T04.

4) Programs cannot be loaded unless the head polarity is correct. Try changing the setting of switch 4 of the system switch to reverse the head polarity if programs cannot be loaded.

5) For any transfer or collation, use the tape recorder that was used for recording. If the tape recorder for transfer or collation is different from that used for recording, no transfer nor collation may be possible.

Chapter 8 Monitor

8.1 General

Although a machine language program is difficult to understand because of the numeric fashion in which data is presented, it has many advantages, e.g., it runs much faster and requires less memory space than a BASIC program. Moreover, machine language makes it possible to develop more hardware-oriented programs, to make fuller use of your computer. You can develop machine language programs by using the monitor commands.

This chapter describes the function and use for each monitor command.

When using a monitor command, note the following points.

* Any monitor command is accepted after the [CR] key is pressed.
* Any command must be input exactly as it is described in this manual. Do not enter spaces in the command line.
* Single-byte data in a monitor command must be specified with a 2-digit hexadecimal number, and 2-byte (address) data must be specified with a 4-digit hexadecimal number. The "0" in the upper digit must not be omitted.
* Filename characters exceeding the limit are ignored.
* The entire memory space can be accessed by monitor commands. However, remember that the presence of even a single error in a program is likely to result in the destruction of all data stored in your MZ-800.

8.2 ROM Monitor and BASIC Monitor

The MZ-800 is provided with two types of monitors: a ROM monitor and a BASIC monitor. The ROM monitor resides (is located) in ROM, while the BASIC monitor is loaded into RAM when you load the BASIC interpreter.
The difference between the ROM and BASIC monitors is shown below.



8.3 Starting the ROM Monitor

When you turn on the power to the MZ-800, you will see the following screen.

```
Make ready CMT

Please push key

C: Cassette tape
M: Monitor
```

Press the [M] key to start the monitor. The screen will then change and appear as follows.

```
* *  MONITOR 9Z-504M  * *
*
```

The asterisk (*) on the second line is called the monitor prompt, and asks you to enter a monitor command.

The monitor commands are explained in Section 8.4.

To terminate the monitor, turn off the power switch.

8.4 Monitor Commands

L Command

Format

L

Explanation

This command loads a machine language program from the cassette. When "↓PLAY" is displayed on the screen, press the [PLAY] button.

Example

The following example loads a machine language program.

```
*L
↓ PLAY ← Press the [PLAY] button on the data recorder.
```

S Command

Format

S

Explanation

This command saves the specified memory block onto the cassette with specified filename.

Example

The following example saves a machine language program stored in addresses $6000 to $60A3 onto the cassette under the filename "MFILE". The address from which the program is to be executed is $6050.

```
*S [CR]
Filename? MFILE [CR]
Top adrs? 6000 [CR]
End adrs? 60A3 [CR]
Exc adrs? 6050 [CR]
↓ RECORD.PLAY
```

Press the [RECORD] button

M Command

Format M <starting address>

Explanation This command modifies the contents of memory, starting at the specified address.

Example The following example fills addresses $C000 to $C002 with the value $FF and addresses $C010 to $C013 with the value $88.

```
*MC000 [CR]
C000 00 FF
C001 00 FF
C002 00 FF
C003 00 [SHIFT] + [BREAK]
*MC010 [CR]
C010 00 88
C011 00 88
C012 00 88
C013 00 88
C014 00 [SHIFT] + [BREAK]
```

To return to the monitor prompt, press [SHIFT] + [BREAK].

J Command

Format J<address>

Explanation This command transfers control to the specified address, by loading the <address> into the program counter of the CPU.

Example The following example transfers control to address $1200.

```
*J1200 [CR]
```

G Command

Format G<address>

Explanation This command calls the specified address.

Example The following example calls address $1200.

```
*G1200 [CR]
```

D Command

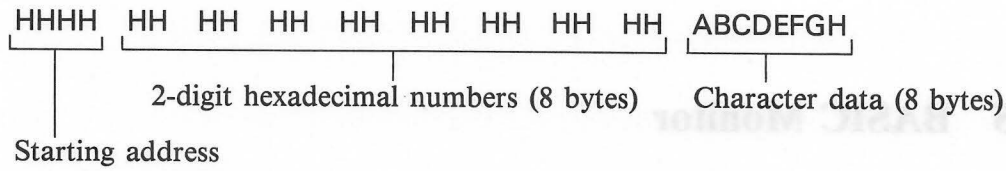
Format

D<starting address><end address>

Explanation

This command dumps the contents of the specified memory area.

When the <end address> is omitted, 160 bytes from the <starting address> are displayed.
The dislay format is as follows:



To modify the memory contents, move the cursor to the data to be modified, type in the new data and press the CR key.

Note:
The last eight characters indicate the ASCII codes corresponding to eight hexadecimal numbers. A contorl code is represented by a period (.). To stop the screen display, press the space bar; to return to the monitor prompt, press BREAK while holding down the SHIFT key.

Example

The following example dumps the contents of addresses $C000 to $C700.

```
*DC000C700 CR
```

V Command

Format

V

Explanation

This command verifies data saved on the cassette, or checks whether the data saved on the tape and the data in memory are identical.
When no incorrect data is detected, the message "OK!" is displayed. If one or more bytes that do not match are detected, the message "CHECK SUM ERROR" is displayed.

Example

The following example verifies the data of file "MFILE" which has been previously saved with the S command.

```
*VMFILE CR
↓PLAY ←——— Press the PLAY button.
```

B Command

Format	**B**
Explanation	This command specifies that the buzzer in the MZ-800 sounds every time a key is pressed. If the B command is entered again, the buzzer toggles off and no longer sounds.
Example	*B CR

8.5 BASIC Monitor

When the BASIC interpreter is used, the BASIC monitor can be used instead of the ROM monitor. To call the BASIC monitor, key in the BASIC BYE command. After the prompt "*" is displayed, key in a BASIC monitor command.
The BASIC monitor uses memory area $FF00 to $FFFF as its stack area.
All variables for BASIC programs are not changed when the BASIC monitor is called, but they can be changed by monitor commands.

8.6 BASIC Monitor Commands

P Command (Print switch)

Format

P

Explanation

This command outputs the data produced by the D or F command to the printer or screen depending on whether the current operating mode is the printer mode or the screen mode. When the BASIC monitor is started, the screen mode becomes valid. The mode is changed each time the P command is entered.
In the printer mode, if no printer is connected or the printer is off-line, the monitor prompt (*) is displayed preceded by the message "ERR?".
Check the printer or key in the P command to enter the screen mode.

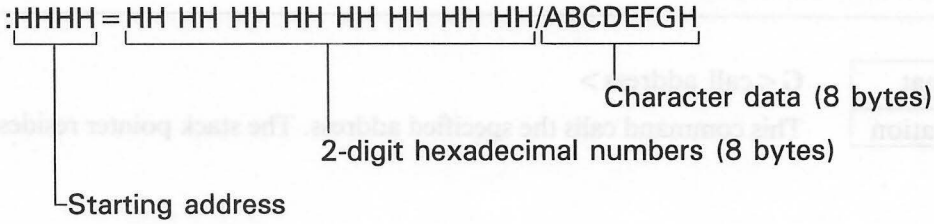
D Command (Dump)

Format

D<starting address><end address>

Explanation

This command displays the contents of the main memory. When the end address is omitted, the 128 bytes following the starting address are displayed. When the starting address is omitted, the 128 bytes following the last end address are displayed. The display format is as follows:



To modify the memory contents, move the cursor to the data to be modified, type in a 2-digit hexadecimal number or character preceded by a semicolon and press the CR key.

Note:
The last eight characters indicate the ASCII code equivalents to the eight hexadecimal numbers. Control codes are represented by periods (.). To stop the screen display, press the BREAK key, and to return to the monitor prompt, press BREAK while holding down the SHIFT key.

M Command (Memory set)

Format M<starting address>

Explanation This command modifies the contents of the main memory. When the <starting address> is omitted, modification is made from the address indicated by the current pointer. To return to the monitor prompt, press BREAK while holding down the SHIFT key.
When the M command is entered, the cursor positions itself at the data for the specified address. The address pointer is incremented by the number of data bytes specified. Data may be either a 2-digiht hexadecimal or a character preceded by a semicolon.

F Command (Find)

Format F<starting address><end address><data><data> ...

Explanation This command searches for one or more bytes of data at the specified addresses, and if found, displays the addresses and data with the format shown for the D command. To return to the monitor prompt, press BREAK while holding down the SHIFT key.

G Command (Gosub)

Format G<call address>

Explanation This command calls the specified address. The stack pointer resides at address $FEFF.

T Command (Transfer)

Format T<starting address><end address><destination address>

Explanation This command transfers data from the specified source address to the specified destination address.

S Command (Save)

Format

S<starting address><end address><execution address>:
<device name>:<filename>

Explanation

This command saves data from the specified address onto the specified device. The execution address is the address to which control is to be transferred when the program is loaded by the L command. Filename must be specified after a colon (:).

L Command (Load)

Format

L<starting address>:<device name>:<filename>

Explanation

This command loads the specified file from the specified device. If the <starting address> is omitted, the file is loaded to the same address as that specified when the file was first saved by the S command. If filename is omitted when the device is CMT:, the first file found is loaded. When the SHIFT + BREAK key is pressed or a check sum error occurs during the load operation, the message "ERR?" is displayed, followed by the monitor prompt.

V Command (Verify)

Format

V<filename>

Explanation

This command loads the specified file from the cassette and compares it with the same file still in the main memory. The purpose of this command is to check whether the file was saved onto the cassette correctly.
If an error is detected, the message "ERR?" is displayed.

R Command (Return)

Format

R

Explanation

This command returns control to the program from which the monitor was called. If the stack pointer for the program which called the monitor resides in addresses $FF00 to $FFFF or if no return address is saved in the stack, control cannot be returned by the R command. When this happens, warm start the computer with the G command.

Chapter 9 MZ-700 Mode

9.1 Using MZ-700 Programs

Most of the programs for the SHARP MZ-700 series computer can be run on your MZ-800 computer. However, programs which use joystick MZ-1X03 cannot be used. Please consult your dealer to check whether the MZ-700 programs you already have can be used with the MZ-800.
To run on an MZ-700 program on your MZ-800, you must first place the MZ-800 in the MZ-700 mode. This can be done by switching switch 1 of the system switch on the rear panel ON, then turning on the power to the MZ-800.

MZ-700 BASIC (1Z-013) is recorded on the beginning of the side of the cassette which is labeled "BASIC 1Z-013".

Three BASIC demonstration programs for the MZ-700 are recorded on the tape following MZ-700 BASIC.

These programs can be executed as follows.

After loading BASIC (1Z-013), advance the tape to one of the values indicated below, then input the following.

"OPENING"..... 130
"MUSIC" 170
"COLOR PLOTTER.... 190

RUN "CMT:" [CR]

When " ↓ PLAY" is displayed, press the [PLAY] button.

After the tape stops, press the [STOP] button. To stop the program, press the [SHIFT] and [BREAK] keys at the same time.

9.2 Summary of MZ-700 BASIC Commands and Statements, Functions and Operations

Commands

LOAD	LOAD "ABC"	Loads BASIC text file ABC from the cassette tape into memory.
SAVE	SAVE "E"	Names the BASIC text currently in the text area "E" and writes in to the cassette tape.
RUN	RUN	Executes the program from the heading of the BASIC text currently in the text area. **Note:** At the RUN command, all variables become 0 or null immediately prior to program execution.
	RUN 1000	Executes program from statement number 1000.
MERGE	MERGE "TEST"	Merges program currently in the memory and "TEST" file in the cassette tape.
VERIFY	VERIFY "H"	Compares program text currently in BASIC text area and content of cassette tape file specified by file name "H".
AUTO	AUTO	Automatically generates line numbers 10, 20, 30 ... during text making.
	AUTO 200, 20	Automatically generates 200 220, 240 ... in steps of 20, from statement number 200. AUTO command is released by pressing [SHIFT] + [BREAK] keys.
LIST	LIST	Displays all lists of BASIC text currently in text area.
	LIST−500	Displays list up to statement number 500.
LIST/P	LIST/P	Display list goes to printer. (TEXT MODE)
RENUM	RENUM	Changes statement number of the program.
	RENUM 100	Renumbers all statements beginning with first statement number 100, and in steps of 10.
NEW	NEW	Erases BASIC text currently in text area and clears variable area. Machine language area specified by LIMIT command is not cleared.
CONT	CONT	Continues program execution. In other words, restarts execution from point of interruption by [SHIFT] + [BREAK] keys or STOP statement during program. CONT command becomes invalid when, during a program break, the BASIC text is edited.
BYE	BYE	Moves system control from BASIC to monitor. (The return from monitor to BASIC can be made by monitor command "R".)
KEY LIST	KEY LIST	Lists, on the CRT display, the definition condition of the definable function keys.

File control statements

WOPEN	10 WOPEN "DATA"	Opens a data file on cassette tape prior to writing data to it. This command also assigns name DATA to the data file.
PRINT/T	20 PRINT/T X	Writes data to cassette tape in the same format as it would be displayed by the PRINT statement.
ROPEN	10 ROPEN "DATA"	Searches for data file DATA on cassette tape and opens that file to prepare for reading data from it.
INPUT/T	20 INPUT/T X	Inputs data from a cassette file and passes it to variable X.
CLOSE	30 CLOSE	Closes cassette data files after writing or reading has been completed.

Error processing statements

ON ERROR GOTO	ON ERROR GOTO 1000	If an error occurs during program execution, this is a sentence saying to jump to statement number 1000.
IF ERN	IF ERN=43 THEN 1050	If the error number is 43, this is a command to jump to statement number 1050.
IF ERL	IF ERL=350 THEN 1090	A command to jump to statement number 1090 if the error statement number is 350.
	IF (ERN=43)* (ERL=700) THEN END	A command to finish the program if the error number is 43 and the error statement number is 700. For the BASIC, if an error occurs during the program, the error number and error statement number will be set, respectively, to variables ERN and ERL.
RESUME	650 RESUME	Transfers control once again to the command generating the error.
	700 RESUME NEXT	Transfers control to the command following the command generating the error.
	750 RESUME 400	Transfers control to statement number 400.
	800 RESUME 0	Trasnfers control to the program heading.

Substitution statement

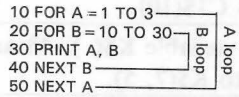
LET	LET A=X+3	Substitutes sum results of numerical variable X and numerical data 3 to numerical variable A. LET can be omitted.

Input/output and colour control statements

COLOR	10 COLOR,,,2	Changes all screen background colour to red.
	20 COLOR 3,2,7	Changes the colour of characters at coordinates (3,2) to white.
	30 COLOR 4,2,4,2	Makes the colour of characters at coordinates (4,2) green, and the background colour red.
PRINT	10 PRINT A	Displays the content of numerical variable A on the CRT display.
	?A$	Displays the content of string variable A$ on the CRT display.
	100 PRINT [6,5] "ABC"	Writes the "ABC" string in yellow on a light blue background.
	110 PRINT [,4] "DEF"	Writes the "DEF" string in yellow on a green background.
	120 PRINT [7,4] "GHI"	Writes the "GHI" string in white on a green background.
	200 PRINT	New line if PRINT only.
PRINT USING	PRINT USING "# # #.# #";A	A designation which lines up decimal point positions by a fixed decimal point display.
INPUT	10 INPUT A	Inputs values relative to variable A from the keyboard.
	20 INPUT A$	Inputs strings relative to string variable A$ from the keyboard.
	30 INPUT "VALUE?";D	Before input from the keyboard, the question string data VALUE? is displayed. The semi-colon is used to separate the string from the variable.
	40 INPUT X, X$, Y, Y$	Numerical variables and string variables can be combined by using the comma (,) to separate them, but it is necessary to match the type of variable at the time of input.
SET	SET 30,15	Illuminates the position of coordinates (30,15).
RESET	RESET 30,15	Erases the position of coordinates (30,15).

GET	10 GET N	Inputs one numerical character from the keyboard relative to numerical variable N. If the key is not pressed at that time, 0 is input.
	20 GET K$	Inputs one string from the keyboard relative to string variable K$. If the key is not pressed at that time, A$ becomes vacant.
READ ~DATA	10 READ A,B,C 1010 DATA 25, −0.5, 500	Numerical data 25, −0.5 and 500 are substituted to, respectively, numerical variables A, B and C by execution of the READ-DATA statements at the left.
	10 READ H$,H, S$,S 30 DATA "HEART", 3 35 DATA "SPADE", 11	The first data of the DATA statement, i.e., string data "HEART", is substituted for the first variable of the READ statement, i.e., for the string variable H$. Next, numerical data 3 is substituted for the second variable H, and read-in continues one after the other.
RESTORE	10 READ A,B,C 20 RESTORE 30 READ D,E 100 DATA 3, 6, 9, 12, 15	In the example at the left, 3, 6 and 9 are respectively substituted for variables A, B and C by the READ statement in statement number 10, but, because the RESTORE statement occurs next, the values next substituted for variables D and E by statement number 30's READ are, respectively, 3 and 6, not 12 and 15.
	700 RESTORE 200	Moves the data read-out pointer in the READ-DATA statement to the heading of the DATA statement in statement number 200.

Loop statements

FOR ~ NEXT	10 FOR A=1 TO 10 20 PRINT A 30 NEXT A	The statement number 10 is a command to change variable A and substitute for values from 1 to 10; the value of the first A becomes 1. Because the value of A is displayed on the CRT screen by statement number 20, the numeral 1 is displayed. Next, the value of A becomes 2 by statement number 30, and this loop is repeated. The loop is repeated in this way until the value of A becomes 10. (At the point when the loop ends, the value 11 is entered to A.)
	10 FOR B = 2 TO 8 STEP 3 20 PRINT B 30 NEXT B	A command to change variable B and substitute for values from 2 to 8 in steps of 3 (statement number 10). It is also possible to make the STEP value negative and make the variable smaller each time.
		An example of an overlay of the FOR~NEXT loops (variables A and B). Note that B loop is placed inside A loop. Nesting of loops (doubling, tripling ...) is possible, but the inner loop must be enclosed within the outer loop. FOR~NEXT nesting must not exceed 15 levels.

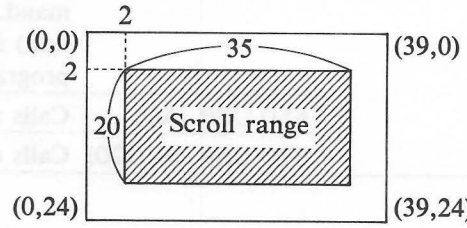
Branch statements

GOTO	100 GOTO 200	Jumps to statement number 200 (=movement of program execution).
GOSUB~ RETURN	100 GOSUB 700 ..... 800 RETURN	Branches to statement number 700 subroutine (calling of subroutine). Ends subroutine execution by RETURN statement, and returns to statement following GOSUB command in the main program.
IF~THEN	10 IF A>20 THEN 200	Jumps to statement number 200 if variable A is larger than 20. Executes next statement if A is 20 or less.
	50 IF B<3 THEN B=B+3	Substitutes B+3 for variable B if variable B is less than 3. Executes next statement if B is 3 or greater.
IF~GOTO	100 IF A>=B GOTO 10	Jumps to statement number 10 if variable A is equal to or greater than variable B. Executes next statement if A is less than B.
IF~GOSUB	30 IF A=B*2 GOSUB 90	Branches to statement number 90 subroutine if value of variable A is equal to twice the value of B. If not, executes next statement. (If there is a multi-statement following a conditional statement, the ON statement is executed when the condition is not reached, but the IF statement moves the execution to the next statement number if the condition is not reached, and the multi-statement is ignored.)
ON~GOTO	50 ON A GOTO 70, 80, 90	Jumps to statement number 70 if variable A is 1, to statement number 80 if it is 2, and to statement number 90 if it is 3. The next statement is executed if A is 0 or 4 or more. The INT function is included in ON, so jumps to statement number 80 if A is 2.7, in the same way as 2.
ON~GOSUB	90 ON A GOSUB 700, 800	Branches to statement number 700 subroutine if variable A is 1, and to statement number 800 if it is 2. The next statement is executed if A is 0 or 3 or more.

Definition statements

DIM	10 DIM A(20)	For one-dimensional numerical array variable A( ), 21 array variables become available, from A(0) to A(20).
	20 DIM B(79,79)	For two-dimensional numerical array variable B( ), 6400 array variables become available, from B(0, 0) to B(79,79).
	30 DIM C1$(10)	For one-dimensional string array variable C1$( ), 11 array variables become available, from C1$(0) to C1$(10).
	40 DIM K$(7,5)	For two-dimensional string array variable K$( ), 48 array variables become available, from K$(0, 0) to K$(7, 5).
DEF FN	100 DEF FNA (X) =X↑2−X 110 DEF FNB (X) =LOG (X)+1 120 DEF FNZ (Y) =LN (Y)	Statement number 100 defines X^2-X to FNA (X), statement number 110 defines $\log_{10}X+1$ to FNB (X), and statement number 120 defines $\log_e Y$ to FNZ (Y). Each function is limited to 1 variable.
DEF KEY	15 DEF KEY(1)= "LIST"+CHR$ (13) 25 DEF KEY(2)= "LOAD:RUN" +CHR$(13)	The DEF KEY statement of statement number 15 defines the function LIST [CR] to function key number 1, and statement number 25 defines the function LOAD:RUN [CR] to function key number 2.

Comment statements and control statements

REM	200 REM JOB1	REM is a comment statement; ignored when program is executed.
STOP	850 STOP	Stops program execution and awaits command. If CONT command given here, program continues.
END	2000 END	Indicates end of program. Executes program end.
CLR	300 CLR	All numerical variables and character variables become 0 or vacant (null); all array variables return to undetermined condition. All DEF FN statements also become invalid.
CURSOR	50 CURSOR 25, 15 60 PRINT "ABC"	Specifies the position by numerals or variables: form 0 to 39 from the left end in the X axis direction , and 0 to 24 from the top end in the Y axis direction. For the example at the left, string "ABC" is displayed from the 26th cursor position from the left end of the screen and the 16th cursor position from the top end.
CONSOLE	10 CONSOLE 0, 25, 0, 40	The scroll range covers the whole screen.
	20 CONSOLE 5, 15	Specifies the scroll range form the 5th line to the 15th line.
	30 CONSOLE 0, 25, 5, 30	Specifies the scroll range from the 5th line to the 30th line.
	40 CONSOLE 0, 10, 0, 10	Specifies the scroll range to a 10×10 range.
	50 CONSOLE 2, 20, 2, 35	Specifies the scroll range to the scroll range shown in the figure below. 
SIZE	? SIZE	Displays the unused size (in bytes) of the BASIC text area.
TI$	100 TI$ = "222030"	Sets the internal clock to 10:20:30 PM. Time data are expressed as a 6-digit figure within quotation marks.

Music control statements

MUSIC **TEMPO**	300 TEMPO 7 310 MUSIC"DE# FGA"	Tempo 7 (fastest speed) is specified by statement number 300. By statement number 310, re mi fa# sol la (midrange) are played at tempo 7. If there is no TEMPO statement, the music is played at the tempo of the default value.
	300 M1$="C3E G+C" 310 M2$="BGD −G" 320 M3$= "C8R5" 330 MUSIC M1$, M2$,M3$	In this example, the melody is substituted to the 3 string variables and the MUSIC command is executed. When the staff notation is used, the notes below are played. Note that, because there is o TEMPO statement, the playing is at the default value tempo. 

Machine language program control statements

INP@	INP@$E8,A	Substitutes data at port number $E8 for variable A.
OUT@	OUT@$E8,A	Outputs variable A to port number $E8.
LIMIT	100 LIMIT 49151	Limits the area used by the BASIC program to the 49151 address (BFFF with hexadecimal notation).
	100 LIMIT A	Limits the area used by the BASIC program to the address of variable A.
	100 LIMIT $BFFF	Limits the area used by the BASIC program to the address BFFF in hexadecimal notation.A hexadecimal notation is indicated by a "$" mark before the notation.
	300 LIMIT MAX	Returns the area used by the BASIC program to the maximum memory.
POKE	120 POKE 49450, 175	Sets data 175 (decimal notation) to the decimal notation address 49450.
	130 POKE AD, DA	Sets the value (0 to 255) indicated by variable DA to the address specified by variable AD.
PEEK	150 A=PEEK (49450)	Changes the data at decimal notation address 49450 to a decimal number, and substitutes for variable A.
	160 B=PEEK (C)	Changes data entered at the decimal notation address specified by variable C to a decimal notation, and substitutes for variable B.
USR	500 USR (49152)	Moves program control to decimal address 49152. This control movement has the same function as the machine language CALL command. As a result, when the RET command (201 at decimal notation) is in the machine language program, returns to the BASIC program.
	550 USR (AD)	Calls the decimal address specified by variable AD.
	570 USR ($C000)	Calls the hexadecimal address C000.

Printer control statement

AXIS		Valid in GRAPH mode.
	30 AXIS 0, −10, 48	Adds a scale of 48 graduations in increments of 10 to the Y-coordinate axis from the current pen position.
	50 ASIX 1, 10, 48	Adds a scale of 48 graduations in increments of 10 to the X-coordinate axis from the current pen position.
CIRCLE		Valid in GRAPH mode.
	50 CIRCLE 0, 0, 240, 0, 360, 0, 2	Draws a circle (radius 240) from coordinates (0,0).
GPRINT		Valid in GRAPH mode.
	30 GPRINT (2,2), "A"	Prints the character A upside down at the size of the 26-digit mode of the TEXT mode.
HSET	30 HSET	Specifies the current pen position to a new starting point. (Valid in GRAPH mode.)
LINE		Valid in GRAPH mode.
	10 LINE% 1, 240, 0, 240, −240, 0, −240, 0, 0	Coordinates (240,0), (240,−240), (0,−240) and (0,0) are connected by a solid line from the current pen position.

MODE	MODE TN	Returns from the GRAPH mode to the TEXT mode (40 characters per line).
	MODE TL	Returns from the GRAPH mode to the TEXT mode (26 characters per line).
	MODE TS	Returns from the GRAPH mode to the TEXT mode (80 characters per line).
	MODE GR	Switches from the TEXT mode to the GRAPH mode (in order to draw graphs and figures).
MOVE		Valid in GRAPH mode.
	10 MOVE 150, 100	Moves the pen upward from the current pen position to coordinates (150, 100).
RMOVE		Valid in GRAPH mode.
	20 RMOVE −240, 240	Moves the pen upward relatively from the current pen position by −240 (X direction) and 240 (Y direction).
PAGE		Valid in TEXT mode.
	10 PAGE 30	Specifies 30 lines per page.
PCOLOR		Valid in both TEXT and GRAPH mode.
	10 PCOLOR 1 20 PRINT/P "ABC"	Prints "ABC" to the plotter printer in blue.
PHOME	PHOME	Moves the pen upward from the current pen position and returns to the starting point. (Valid in GRAPH mode).
PLOT	PLOT ON	Enables use of colour plotter printer as substitution for the display. (Valid in TEXT mode.)
	PLOT OFF	Cancels above function.
PRINT/P		Valid in TEXT mode.
	10 PRINT/P A, A$	Outputs string variable A$ content after the numerical variable A content to printer.
	20 PRINT/P "H"	For form feed of printer.
PRINT/P USING		Outputs format specified data to screen. Format specification is written after the word USING.
	PRINT/P USING "# # # #";A	Numerical variable A contents are output to printer within 4 digits, justified right.
RLINE		Valid in GRAPH mode.
	70 RLINE% 1, 240, 0, −120, −SQ, −120, SQ	Connects specified positions, relatively from current pen position (240,0), (−120, −SQ) and (−120,SQ) by solid line.
SKIP		Valid in TEXT mode.
	10 SKIP 10	Advances the paper 10 lines.
	20 SKIP −10	Rewinds 10 lines.
TEST	TEST	Checks colour specification and ink amount and dryness. (Valid in TEXT mode).

Arithmetic functions

ABS (X)	A=ABS (X)	Assigns the absolute value of variable |X| to variable A. Example: A=ABS (2.9)→A=2.9 A=ABS (−5.5)→A=5.5
SGN (X)	A=SGN (X)	Assigns the numeric sign of variable X to variable A. If the value of X is negative, −1 is assigned to A; if X is 0, 0 is assigned to A; and if X is positive, 1 is assigned to A. $A=\begin{cases}1 & (X>0)\\ 0 & (X=0)\\ -1 & (X<0)\end{cases}$ Example: 1 is assigned to variable A when A=SGN (0.4) is executed.
INT (X)	A=INT (X)	Assigns the greatest integer value to A which is less than or equal to the value of variable X. Examples: A=INT (3.87) →A=3 A=INT (0.6) →A=0 A=INT (−3.87)→A=−4
SIN (X)	A=SIN (X) A=SIN(30*PAI(1)/180)	Assigns the sine of X (where X is in radians) to variable A. If the value of X is in degrees, it must be converted to radians before this function is used to obtain the sine. Since 1 degree equals π/180 radians, the value in radians is obtained by multiplying the number of degrees by PAI(1)/180. For example, 30° = 30*PAI(1)/180 radians. The same applies to the COS, TAN, and ATN functions.
COS (X)	A=COS (X) A=COS (200*PAI(1)/180)	Assigns the cosine of X (where X is in radians) to variable A.
TAN (X)	A=TAN (X) A=TAN(Y*PAI(1)/180)	Assigns the tangent of X (where X is in radians) to variable A.
ATN (X)	A=ATN (X) A=180/PAI(1)*ATN(X)	Assigns the arctangent in radians of X ($\tan^{-1}X$) to variable A. The value returned will be in the range from $-\pi/2$ to $\pi/2$.
SQR (X)	A=SQR (X)	Calculates the square root of X and assigns the result to variable A. X must be a positive number or 0.
EXP (X)	A=EXP (X)	Calculates the value of e^x and assigns the result to variable A.
LOG (X)	A=LOG (X)	Calculates the common logarithm of X ($\log_{10}$ X) and assigns the result to variable A.
LN (X)	A=LN (X)	Calculates the natural logarithm of X ($\log_e$ X) and assigns the result to variable A.
PAI (X)	A=PAI (X)	Assigns the value to variable A which is X times the value of π.
RAD (X)	A=RAD (X)	Converts the value of X (where X is in degrees) to radians and assigns the result to variable A.

String control functions

LEFT$	10 A$=LEFT$ (X$,N)	Substitutes string variable X$ (from beginning to Nth character) for string variable A$. It doesn't matter whether N is a constant, variable or numerical formula.
MID$	20 B$=MID$(X$, M, N)	Substitutes string variable X$ (from Mth character to N charater) for string variable B$.
RIGHT$	30 C$=RIGHT$ (X$, N)	Substitutes string variable X$ (from end to N character) for string variable C$.
SPC	40 D$=SPC (N)	Substitutes N number of spaces for string variable D$.
CHR$	60 F$=CHR$(A)	Converse to the ASC function, substitutes ASCII code characters which are equivalent to the value of real number A for string variable F$. It doesn't matter whether A is a constant, variable or numerical formula.
ASC	70 A=ASC (X$)	Substitutes the value of the ASCII code of the first character of string variable X$ for variable A.
STR$	80 N$=STR$ (1)	Converts to the VAL variable, substitutes the numerical variable I as if it were a string for string variable N$.
VAL	90 I=VAL (N$)	Substitutes the numerical string of string variable N$ as if it were a number for variable I.
LEN	100 LX=LEN (X$)	Substitutes the character length (character number) of string variable X$ for variable LX.
	110 LS=LEN (X$+Y$)	Substitutes the sum of the character length of string variables X$ and Y$ for variable LS.

Tab function

TAB	10 PRINT TAB (X);A	Displays the value of variable A at the X + 1 character position counting from the left edge of the screen.

Arithmetic operations

The calculation priority is of white figures on dark background at left side, but the calculation of figures in parentheses ( ) has even higher priority.

↑	10 A=X↑Y (power)	Substitutes the X↑Y calculation result for variable A. (Note, however, that an error occurs if Y is not an integral number when X is a negative number at X↑Y.)
−	10 A=−B (minus sign)	0−B is a subtraction; note that the "−" of −B is a minus sign.
*	10 A=X*Y (multiplication)	Substitutes the multiplication result of X and Y for variable A.
/	10 A=X/Y (division)	Substitutes the division result of X and Y for variable A.
+	10 A=X+Y (addition)	Substitutes the addition result of X and Y for variable A.
−	10 A=X−Y (subtraction)	Substitutes the subtraction result of X and Y for variable A.

Comparison logic operators

=	10 IF A=X THEN	If variables A and X are equal, executes commands from THEN onward.
	20 IF A$="XYZ" THEN	If string variable A$ content is string XYZ, executes commands from THEN onward.
>	10 IF A>X THEN	If variable A is greater than X, executes commands from THEN onward.
<	10 IF A<X THEN	If variable A is smaller than X, executes commands from THEN onward.
<>or><	10 IF A<>F THEN	If variable A and X are not equal, executes commands from THEN onward.
>= or =>	10 IF A>=X THEN	If variable A is greater than or equal to X, executes commands from THEN onward.
<= or =<	10 IF A<=X THEN	If variable A is smaller than or equal to X, executes commands from THEN onward.
*	40 IF (A>X)*(B>Y) THEN	If variable A is greater than X and variable B is greater than Y, executes commands from THEN onward.
+	50 IF (A>X)+(B>Y) THEN	If variable A is greater than X or variable B is greater than Y, executes commands from THEN onward.

Other symbols

?	200 ?"A+B=";A+B 210 PRINT "A+B=";A+B	Can be used instead of PRINT. Consequently, statement number 200 and 210 are the same.
:	220 A=X:B=X↑2:?A,B	A symbol to express punctuation of the command statement; used in multiple commands. There are 3 command statements used in the statement number 220 multiple command.
;	230 PRINT"AB";"CD";"EF"	Executes PRINT continuously. As a result line number 230, "ABCDEF" is displayed on the screen continuously, with no space.
	240 INPUT"X=";X$	Displays "X=" on screen; awaits data key input of string variable X$.
,	250 PRINT"AB","CD","E"	Executes PRINT with tabulation. For statement number 250, first AB is displayed on the screen, then CD is displayed in the position 10 characters to the right of A, and then E is displayed in the position 10 characters to the right of C.
	300 DIM A(20),B$(3,6)	An example used in punctuation of a variable.
" "	330 B$="MZ-700"	" indicates a string content
$	340 C$="ABC"+CHR$(3)	Indicates a string variable.
	500 LIMIT $BFFF	Indicates hexadecimal number.
π	550 S=SIN (X*π/180)	The approximate value of pi (3.1415927) is expressed by π.

Appendixes

Appendix A Display Control in the MZ-800 Mode

(1) Graphics memory

The standard MZ-800 supports a display screen of 320 × 200 dots in 4 colours selected from a possible 16 colours, or a monochrome display screen of 640 × 200 dots.
By installing the optional graphics memory (MZ-1R25), the display capability is improved so that a display screen of 320 × 200 dots can be displayed in 16 colours or a screen of 640 × 200 dots can be displayed in 4 colours selected from a possible 16 colours.

(2) 40-column mode and 80-column mode (Character display)

The number of character columns per line can be switched between 40 and 80 characters with the INIT command.

(3) Display modes (Graphics display)

The resolution and number of colours which can be displayed at any one time differs according to the display mode. The MZ-800's display modes are as follows.

Mode	Resolution	Characters per line	Colours
1	320 × 200 dots	40	4 colours
2	320 × 200 dots		16 colours
3	640 × 200 dots	80	Foreground and background colours
4	640 × 200 dots		4 colours

Modes 2 and 4 can be used only when the optional graphic memory is installed. The graphics display mode is set by the Mn parameter of the INIT command.

For exmaple:

INIT "CRT:M1" ..... Sets mode 1.
INIT "CRT:M2" ..... Sets mode 2.

(4) Colour palette

The colours which can be displayed at one time are selected from the colour palette. The colour palette allows the selection of 4 colours from a possible 16 colours. The 16 colours which can be displayed are listed below along with their colour codes.

Colour code	Colour
0	Black
1	Blue
2	Red
3	Magenta
4	Green
5	Cyan
6	Yellow
7	White
8	Gray
9	Light blue
10	Light red
11	Light magenta
12	Light green
13	Light cyan
14	Light yellow
15	Light white (high brightness white)

In mode 1 or mode 4, palette codes 0 to 3 are used. The initial settings of colour code assignments to the palette codes are as shown below.

Palette code	Colour code (colour)
0	0 (black)
1	1 (blue)
2	2 (red)
3	15 (light white)

In mode 3, palette codes 0 and 1 are used and the initial settings of colour code assignments are as follows.

Palette code	Colour code (colour)
0	0 (black)
1	15 (light white)

Colour code assignments to the palette codes can be changed with the PAL command.

Ex)
PAL 0,4..... Assigns colour code 4 (green) to palette code 0.
PAL 2,7..... Assigns colour code 7 (white) to palette code 2.

(5) Palette usage in mode 2

In mode 2, the initial colour code assignments to the palette codes are as follows.

Palette code	Colour code (colour)
0	0 (black)
1	1 (blue)
2	2 (red)
3	3 (magenta)
4	4 (green)
5	5 (cyan)
6	6 (yellow)
7	7 (white)
8	8 (gray)
9	9 (light blue)
10	10 (light red)
11	11 (light magenta)
12	12 (light green)
13	13 (light cyan)
14	14 (light yellow)
15	15 (light white)

Use of the palette in mode 2 is more complicated. In mode 2, palette codes 0 to 15 are used and they are divided into four blocks as follows.

Palette block No.	0	1	2	3
Palette code	0 to 3	4 to 7	8 to 11	12 to 15

The initial setting of the palette block number is 0 and the initial settings of the colour code assignments to the palette codes are as follows.

Palette code	Colour code (colour)
0	0 (black)
1	1 (blue)
2	2 (red)
3	3 (magenta)

The palette block number can be changed by the INIT command.

Ex)

INIT "CRT:B1" — Changes the palette block number to 1.

The numbers belonging to the current palette block can be used as the palette codes in commands and statements. Some commands and statements have a parameter which specifies a palette code or colour code. The numbers belonging to the current palette block number are recognized as the palette codes, while the other numbers are recognized as the colour codes.
The following example will help you understand the above explanation.

```
10 INIT "CRT:M2,B1"  — Mode 2, palette block No. = 1
20 PAL 4,12          — Assigns colour code 12 to palette code 4.
30 PAL 5,10          — Assigns colour code 10 to palette code 5.
40 PAL 6,8           — Assigns colour code 8 to palette code 6.
50 PAL 7,6           — Assigns colour code 6 to palette code 7.
```

After executing the above program,

```
LINE [5,0] 10,20,100,50
```

draws a line in light red. In this case, the first parameter is recognized as a palette code.

```
LINE [1,0] 10,20,100,50
```

Draws the same line in blue. In this case, the first parameter is recognized as a colour code.

```
INIT "CRT:B0"
```

If the above command is executed after execution of the above program, the result is different.

```
LINE [5,0] 10,20,100,50
```

Draws the line in cyan. In this case, the first parameter is recognized as a colour code.

```
LINE [1,0] 10,20,100,50
```

Draws the line in blue. In this case, the first parameter is recognized as a palette code. When INIT "CRT:B0" is executed, the palette codes which can be set are changed as shown below. The initial settings of the colour code assignments are assumed.

Palette code		Palette code
5		0
6	INIT "CRT:B0"	1
7	⟶	2
8		3

(6) Restoring initial settings

Executing the INIT statement to set a new display mode restores the initial settings of the colour code. Executing the INIT statement to set a new palette block in the 16-colour mode also restores the initial settings.

(7) Logical summing of colours

Some graphic statements such as COLOR, SET and LINE use the "mode" parameter.

When the mode parameter is specified as 0, the resultant colour is specified by the palette code parameter.

When the mode parameter is specified as 1, a logical OR operation is performed between the existing palette code for a dot (on the screen) and the new specified palette code for the same dot, to produce the resultant colour. For example, if the existing palette code for a dot at (50,50) is 2 and the new specified palette code is 1, the resultant palette code is 3.

0010 (binary for 2) OR 0001 (binary for 1) → 0011 (binary for 3)

A table of codes logically ORed is shown below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15
2	2	3	2	3	6	7	6	7	10	11	10	11	14	15	14	15
3	3	3	3	3	7	7	7	7	11	11	11	11	15	15	15	15
4	4	5	6	7	4	5	6	7	12	13	14	15	12	13	14	15
5	5	5	7	7	5	5	7	7	13	13	15	15	13	13	15	15
6	6	7	6	7	6	7	6	7	14	15	14	15	14	15	14	15
7	7	7	7	7	7	7	7	7	15	15	15	15	15	15	15	15
8	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
9	9	9	11	11	13	13	15	15	9	9	11	11	13	13	15	15
10	10	11	10	11	14	15	14	15	10	11	10	11	14	15	14	15
11	11	11	11	11	15	15	15	15	11	11	11	11	15	15	15	15
12	12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
13	13	13	15	15	13	13	15	15	13	13	15	15	13	13	15	15
14	14	15	14	15	14	15	14	15	14	15	14	15	14	15	14	15
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

This logical operation does not however apply to the RESET and BLINE statements. For these two statements, the resultant colour is specified by the colour code given by
{(No. of colours which can be specified in the current colour display mode) − 1 − (specified palette code)} when the mode parameter is 0. For example, when the colour display mode is mode 1 in which a maximum of 4 colours can be specified, specifying colour palette code 1 results in colour palette code 2 as follows.

4 − 1 − 1 = 2

When the mode parameter is 1, the resultant palette code is given by the logical OR of the previous palette code and **{(No. of colours which can be specified in the current colour display mode) minus 1 minus (specified palette code)}**. For example, specifying palette code 1 when the previous palette code is 2 results in palette code 2 as follows.

2 OR 2 → 2

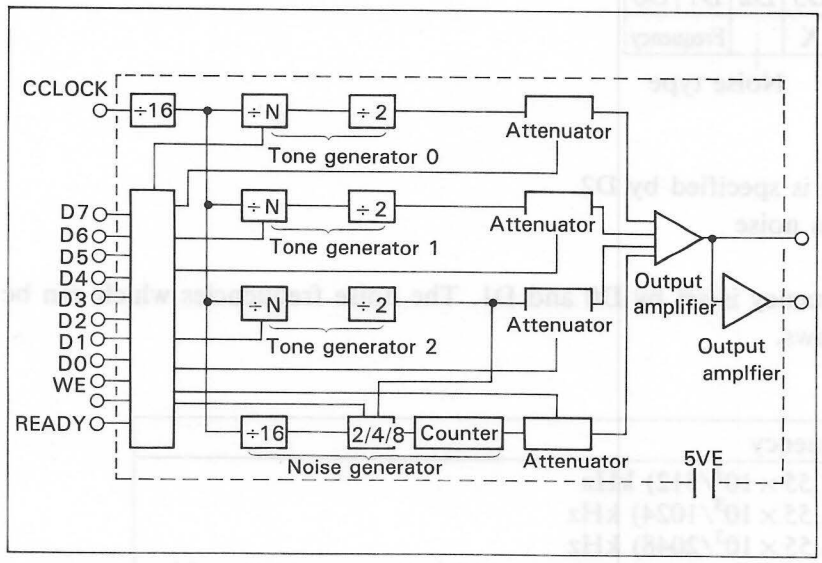
Appendix B Programmable Sound Generator

The MZ-800 has a built-in programmable sound generator (PSG) which makes it possible to generate 3-tone chords over 6 octaves. The PSG is an SN76489AN IC.
The PSG can be controlled by sending data to I/O port address $F2.

(1) Description of the PSG

The SN76489AN IC has eight internal registers, and controls three tone generators and one noise generator.

Block diagram



The internal registers required can be selected by setting bits D4 to D6 in the 1st byte of the output data. The function of each register is shown below.

D6	D5	D4	Function
0	0	0	Frequency of tone 0
0	0	1	Sound volume of tone 0
0	1	0	Frequency of tone 1
0	1	1	Sound volume of tone 1
1	0	0	Frequency of tone 2
1	0	1	Sound volume of tone 2
1	1	0	Noise control
1	1	1	Noise volume

(2) Setting the tone frequency

The tone frequency is set with the following 2-bytes of data.

1st byte

D7	D6	D5	D4	D3	D2	D1	D0
1	Reg. select			Frequency 1 (lower four bits)			

2nd byte

D7	D6	D5	D4	D3	D2	D1	D0
0	X	Frequency (high six bits)					

Frequency = $(3.55 \times 10^3)/(32 \times n)$ kHz

n is a 10-bit binary number represented by D0 to D3 of the 1st byte and D0 to D5 of the 2nd byte.

(3) Noise generation

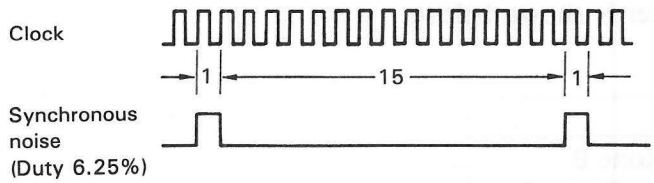
Synchronous noise or white noise can be generated by outputting the following 1-byte data to the I/O port.

D7	D6	D5	D4	D3	D2	D1	D0
1	Reg. select			X		Frequency	

Noise type

* The noise type is specified by D2.
  0: Synchronous noise
  1: White noise
* The noise frequency is set by D0 and D1. The noise frequencies which can be set by D0 and D1 are as follows.

D1	D0	Frequency
0	0	6.93 ($=3.55\times10^3/512$) kHz
0	1	3.47 ($=3.55\times10^3/1024$) kHz
1	0	1.73 ($=3.55\times10^3/2048$) kHz
1	1	Same as that for the sound generated by tone generator #3.



* When white noise is specified, the output waveform is composed of random patterns and a sound with virtually the same spectrum as white noise is generated.
* When noise modes 0, 1, 1 are specified, the frequency is given by the following formula and a low pulse tone can be generated.

$$f = N/32 \times n \times 16$$

* If you specify tone 2 output in the noise mode specification, you must turn the output of tone 2 off.

(4) Setting tone volume

D7	D6	D5	D4	D3	D2	D1	D0
1	Reg. select			Attenuation			

The tone volume is altered with attenuation.

D3	D2	D1	D0	Attenuation (dB)
0	0	0	0	0
0	0	0	1	2
0	0	1	0	4
0	0	1	1	6
0	1	0	0	8
0	1	0	1	10
0	1	1	0	12
0	1	1	1	14

D3	D2	D1	D0	Attenuation (dB)
1	0	0	0	16
1	0	0	1	18
1	0	1	0	20
1	0	1	1	22
1	1	0	0	24
1	1	0	1	26
1	1	1	0	28
1	1	1	1	OFF

Appendix C Reserved Words

ABS
AND
ASC
ATN
AUTO
AXIS
BLINE
BOOT
BOX
BYE
CHAIN
CHR$
CIRCLE
CLOSE #
CLR
CLS
COLOR
CONSOLE
CONT
COS
CSRH
CSRV
CURSOR
DATA
DEF
DEFAULT
DELETE
DIM
DIR
ELSE
END
EOF
ERL
ERN
ERROR
EXP
FN
FOR
GET
GOSUB
GOTO
GPRINT
HCOPY
HSET
IF
INIT
INP@
INPUT
INPUT #
INT
KEY
KEYLIST
KILL #
LABEL
LEFT$
LEN
LET
LIMIT
LINE
LIST
LIST/P
LN
LOAD
LOG
MERGE
MID$
MUSIC
NEW
NEXT
NOISE

NOT
OFF
ON
OR
OUT@
PAGE
PAI
PAINT
PAL
PATTERN
PCIRCLE
PCOLOR
PEEK
PHOME
PLINE
PLOT
PMODE
PMOVE
POINT
POKE
POSH
POSITION
POSV
PRINT
PRINT#
PSKIP
PTEST
RAD
READ
REM
RENAME
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RLINE
RMOVE
RND
ROPEN#
RUN
SAVE
SEARCH
SEARCH/P
SET
SGN
SIN
SIZE
SOUND
SPC
SQR
STICK
STOP
STR$
STRIG
SYMBOL
TAB
TAN
TEMPO
THEN
TI$
TROFF
TRON
USING
USR
VAL
VERIFY
WAIT
WOPEN#
XOR

Appendix D Console Control Codes

If the MZ-800's character set, some of the codes are used to control the operation of the computer as shown below. These control codes can be input to the computer through the keyboard or by using the PRINT statement (indicated with an asterisk (*) in the code column).

Control code table

Code (Dec.)	Key operation	Function
5	CTRL + E	Causes the character keys to input lowercase letters.
6	CTRL + F	Causes the character keys to input uppercase letters.
13 (*)	CTRL + M	Causes a carriage return. CR
16 (*)	CTRL + P	Deletes the character at the position to the left of the cursor position. DEL
17 (*)	CTRL + Q	Moves the cursor down one line. ↓
18 (*)	CTRL + R	Moves the cursor up one line. ↑
19 (*)	CTRL + S	Moves the cursor right one character position. →
20 (*)	CTRL + T	Moves the cursor left one character position. ←
21 (*)	CTRL + U	Moves the cursor to the home position. HOME
22 (*)	CTRL + V	Clears the screen. CLR
23 (*)	CTRL + W	Places the keyboard in the graphics mode. GRAPH
24 (*)	CTRL + X	Inserts a space at the cursor position. INST
25 (*)	CTRL + Y	Places the keyboard in the normal mode. ALPHA

Appendix E Restrictions on Using File I/O Commands and Statements

The file I/O commands and statements cannot be used for all file devices. The table below shows the restrictions on some of these devices.

Command/statement \ Device	CMT: (Data recorder)	RAM: (RAM file board)	CRT: (Display)	LPT: (Printer)	RS1: and RS2: (RS-232C)
INIT	×	○	○	○	○
DEFAULT	○	○	○	○	○
DIR	×	○	×	×	×
RUN	○	○	×	×	×
LOAD	○	○	×	×	×
SAVE	○	○	×	×	×
DELETE	×	○	×	×	×
RENAME	×	○	×	×	×
CHAIN	○	○	×	×	×
MERGE	○	○	×	×	×
WOPEN#	○	○	×	×	○
PRINT#	○	○	×	×	○
ROPEN#	○	○	×	×	○
INPUT#	○	○	×	×	○
CLOSE#	○	○	×	×	○
KILL#	○	○	×	×	○

○: Can be used.
×: Cannot be used.

Further, for CMT:, and RS1: and RS2, only one file can be opened at any one time.

Appendix F Monitor Subroutines

The following subroutines are used by the ROM Monitor (9Z-504M). Each subroutine name symbolically represents the function of the corresponding subroutine. These subroutines can be called from user programs.
Registers saved are those whose contents are restored when control is returned to the calling program. The contents of other registers are changed by execution of the subroutine.

Name and entry point (hex.)	Function	Registers saved
CALL LETNL (0006)	Moves the cursor to the beginning of the next line.	All except AF
CALL PRNTS (000C)	Displays a space at the cursor position.	All except AF
CALL PRNTS (0012)	Displays the character corresponding to the ASCII code stored in the ACC at the cursor position. See Appendix J for the ASCII codes. No character is displayed when code 0D (carriage return) or codes 11 to 16 (the cursor control codes) are entered, but the corresponding function is still performed (a carriage return for 0D and cursor movement for 11 to 16).	All except AF
CALL MSG (0015)	Displays a message, starting at the position of the cursor. The starting address of the area in which the message is stored must be loaded into the DE register before calling this subroutine, and the message must end with a carriage return code (0D). The carriage return is not executed. The cursor is moved if any cursor control codes (11 to 16) are included in the message.	All registers
CALL BELL (003E)	Briefly sounds tone of la (about 880 Hz).	All except AF
CALL MELDY (0030)	Plays a tune according to the music data stored in the memory area starting at the address in the DE register. The music data must be in the same format as that for the MUSIC statement of the BASIC, and must end with 0D or C8. When the tune is completed, control is returned to the calling program with the C flag set to 0. When play is interrupted with the [BREAK] key. Control is returned with the C flag set to 1.	All except AF
CALL XTEMP (0041)	Sets the music tempo according to the tempo data stored in the accumulator (ACC). ACC ← 01 Slowest speed ACC ← 04 Middle speed ACC ← 07 Highest speed Note that the data in the accumulator is not the ASCII codes for 1 to 7 but the binary codes.	All registers
CALL MSTA (0044)	Generates a continuous sound of the specified frequency. The frequency is given by the following equation freq. = 895 kHz/nn'. Here, nn' is a 2-byte number stored in addresses 11A1 and 11A2 (n in 11A2 and n' in 11A1)	BC and DE

<table>
<tr><th>Name and entry point (hex.)</th><th>Function</th><th>Registers saved</th></tr>
<tr><td>CALL MSTP
(0047)</td><td>Stops the sound generated with the CALL MSTA subroutine.</td><td>All except AF</td></tr>
<tr><td>CALL TIMST
(0033)</td><td>Sets and starts the built-in clock. The registers must be set as follows before this routine is called.
ACC ← 0 (AM), ACC ← 1 (PM)
DE ← 4-digit hexadecimal number representing the time in seconds.</td><td>All except AF</td></tr>
<tr><td>CALL TIMRD
(003B)</td><td>Reads the built-in clock and returns the time as follows.
ACC ← 0 (AM), ACC ← 1 (PM)
DE ← 4-digit hexadecimal number representing the time in seconds.</td><td>All except AF and DE</td></tr>
<tr><td>CALL BRKEY
(001E)</td><td>Checks whether the [SHIFT] and [BREAK] keys are both being pressed. The Z flag is set when they are being pressed simultaneously; otherwise, it is reset.</td><td>All except AF</td></tr>
<tr><td>CALL GETL
(0003)</td><td>Reads one line of data from the keyboard and stores it in the memory area starting at the address in the DE register. This routine stops reading data when the [CR] key is pressed, then adds a carraige return code (0D) to the end of the data read.
A maximum of 80 characters (including the carriage return code) can be enered in one line.
Characters keyed in are echoed back to the display. Cursor control codes can be included in the line.
When the [SHIFT] and [BREAK] keys are pressed simultaneously, the BREAK code is stored at the address indicated by the DE register and a carriage return code is sotored in the following address.</td><td>All registers</td></tr>
<tr><td>CALL GETKY
(001B)

Special key read with GETKY</td><td>Reads a character code (ASCII) from the keyboard.
If no key is pressed, control is returned to the calling program with 00 set in ACC.
No provision is made to avoid data read errors due to key bounce, and characters entered are not echoed back to the display.
When any of the special keys (such as [DEL] or [CR]) are pressed, this subroutine returns a code to the ACC which is different to the corresponding ASCII code as shown below. Here, display codes are used to address characters stored in the character generator, and are different from the ASCII codes.
<table>
<tr><th>Special key</th><th>Code loaded in ACC</th><th>Display code</th></tr>
<tr><td>[DEL]</td><td>60</td><td>C7</td></tr>
<tr><td>[INST]</td><td>61</td><td>C8</td></tr>
<tr><td>[ALPHA]</td><td>62</td><td>C9</td></tr>
<tr><td>[BREAK]</td><td>64</td><td>CB</td></tr>
<tr><td>[CR]</td><td>66</td><td>CD</td></tr>
<tr><td>[↓]</td><td>11</td><td>C1</td></tr>
<tr><td>[↑]</td><td>12</td><td>C2</td></tr>
<tr><td>[→]</td><td>13</td><td>C3</td></tr>
<tr><td>[←]</td><td>14</td><td>C4</td></tr>
<tr><td>[HOME]</td><td>15</td><td>C5</td></tr>
<tr><td>[CLR]</td><td>16</td><td>C6</td></tr>
</table></td><td>All except AF</td></tr>
</table>

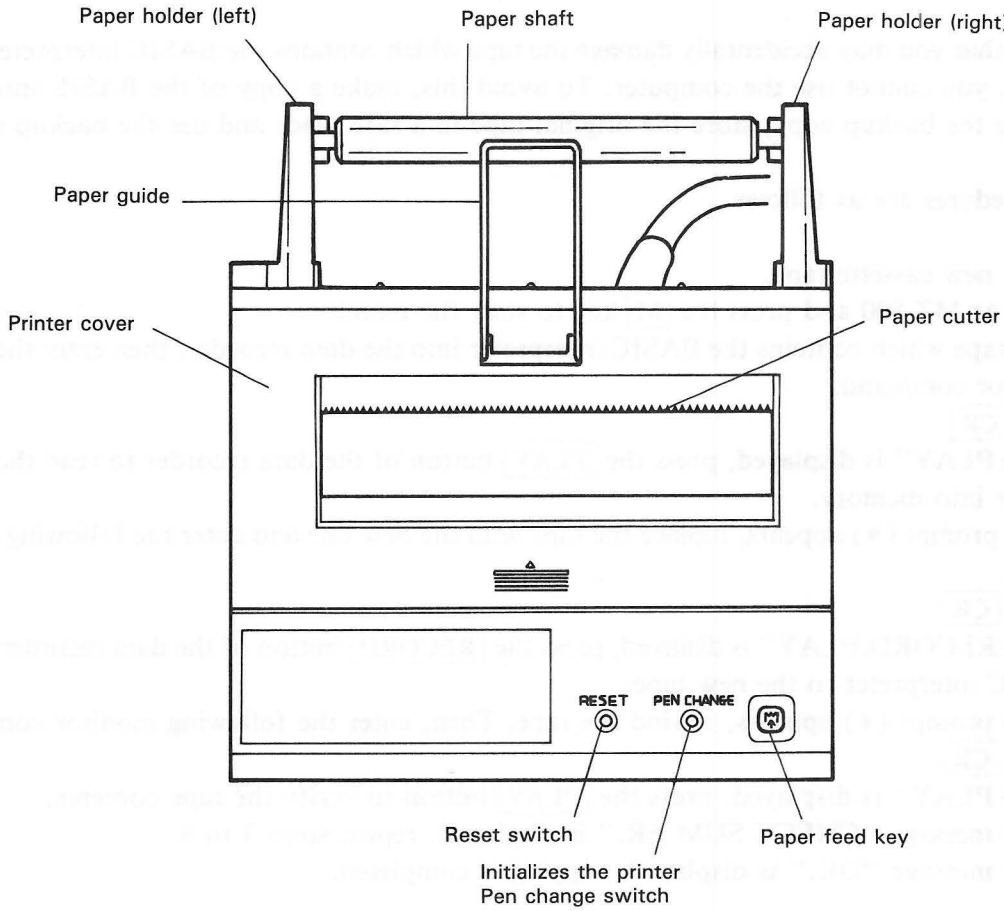
<table>
<tr><th>Name and entry point (hex.)</th><th>Function</th><th>Registers saved</th></tr>
<tr><td>CALL ASC
(03DA)</td><td>Loads the ASCII character corresponding to the hexadecimal number represented by the lower 4 bits of data in ACC.</td><td>All except AF</td></tr>
<tr><td>CALL HEX
(03F9)</td><td>Converts the 8 data bits stored in the ACC into a hexadecimla number (assuming that the data is an ASCII character), then loads the hexadecimal number in the lower 4 bits of ACC. The C flag is set to 0 when a hexadecimal number is loaded in ACC; otherwise, it is set to 1.</td><td>All except AF</td></tr>
<tr><td>CALL HLHEX
(0410)</td><td>Converts a string of 4 ASCII characters into a hexadecimal number and loads it in the HL register. The call and return conditions are as follows.
DE ← Starting address of the memory area which contains the ASCII character string.
(e.g., “3” “1” “A” “5”)
CALL HLHEX ↑—DE
CF = 0 HL←hexadecimal number (e.g., HL = 31A5$_H$)
CF = 1 The contents of HL are not guarenteed.</td><td>All except AF and HL</td></tr>
<tr><td>CALL 2HEX
(041F)</td><td>Converts a string of 2 ASCII characters into a hexadecimal number and loads it into the ACC. The call and return conditions are as follows.
DE ← Starting address of the memory area which contains the ASCII character string.
(e.g., “3” “A”)
CALL 2HEX ↑—DE
CF = 0 ACC←hexadecimal number (e.g., ACC = 3A$_H$)
CF = 1 The contents of the ACC are not guaranteed.</td><td>All except AF and DE</td></tr>
<tr><td>CALL ??KEY
(09B3)</td><td>Blinks the cursor to prompt for key input. When a key is pressed, the corresponding display code is loaded into the ACC and control is returned to the calling program.</td><td>All except AF</td></tr>
<tr><td>CALL ?ADCN
(0BB9)</td><td>Converts ASCII codes into display codes. The call and return conditions are as follows.
ACC ← ASCII code
CALL ?ADCN
ACC ← Display code</td><td>All except AF</td></tr>
<tr><td>CALL ?DACN
(0BCE)</td><td>Converts display codes into ASCII codes. The call and return conditions are as follows.
ACC ← Display codes
CALL ?DACN
ACC ← ASCII code</td><td>All except AF</td></tr>
<tr><td>CALL ?BLNK
(0DA6)</td><td>Detects the vertical blanking period. Control is returned to the calling program when the vertical blanking period is entered.</td><td>All registers</td></tr>
<tr><td>CALL ?DPCT
(0DDC)</td><td>Controls display as follows.
<table>
<tr><th>ACC</th><th>Control</th><th>ACC</th><th>Control</th></tr>
<tr><td>C0$_H$</td><td>Scrolling</td><td>C6$_H$</td><td>Same as the [CLR] key.</td></tr>
<tr><td>C1$_H$</td><td>Same as the [↓] key.</td><td>C7$_H$</td><td>Same as the [DEL] key.</td></tr>
<tr><td>C2$_H$</td><td>Same as the [↑] key.</td><td>C8$_H$</td><td>Same as the [INST] key.</td></tr>
<tr><td>C3$_H$</td><td>Same as the [→] key.</td><td>C9$_H$</td><td>Same as the [ALPHA] key.</td></tr>
<tr><td>C4$_H$</td><td>Same as the [←] key.</td><td></td><td></td></tr>
<tr><td>C5$_H$</td><td>Same as the [HOME] key.</td><td>CD$_H$</td><td>Same as the [CR] key.</td></tr>
</table></td><td>All registers</td></tr>
<tr><td>CALL ?POINT
(0FB1)</td><td>Loads the current cursor location into the HL register. The return conditions are as follows.
CALL ?POINT
HL ← Cursor location (binary)</td><td>All except AF and HL</td></tr>
</table>

Appendix G Making Backup Copy of the BASIC Interpreter

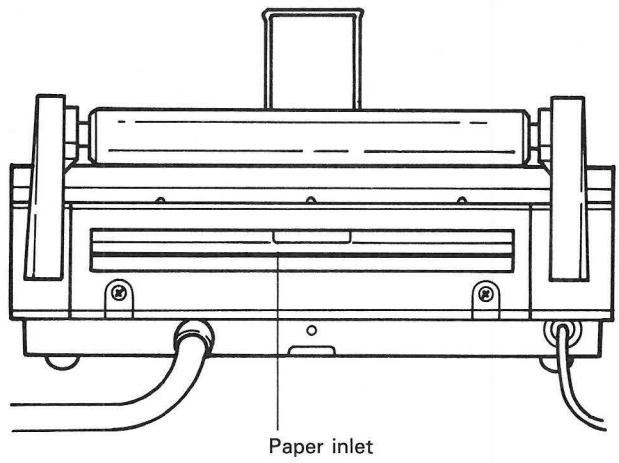
It is possible that you may accidentally damage the tape which contains the BASIC interpreter. When this happens, you cannot use the computer. To avoid this, make a copy of the BASIC interpreter. After making the backup copy, store the original tape in a safe place and use the backup copy for daily use.
Backup procedures are as follows.

1) Prepare a new cassette tape.
2) Turn on the MZ-800 and press the [M] key to start the monitor.
3) Load the tape which contains the BASIC interpreter into the data recorder, then enter the following monitor command.
   *GE807 [CR]
4) When "↓PLAY" is displayed, press the [PLAY] button of the data recorder to read the BASIC interpreter into memory.
5) When the prompt (*) appears, replace the tape with the new one and enter the following monitor command.
   *GE80A [CR]
6) When "↓RECORD.PLAY" is dislayed, press the [RECORD] button of the data recorder to write the BASIC interpreter to the new tape.
7) When the prompt (*) appears, rewind the tape. Then, enter the following monitor command.
   *GE80D [CR]
8) When "↓PLAY" is displayed, press the [PLAY] button to verify the tape contents.
9) When the message "CHECK SUM ER." is displayed, repeat steps 3 to 8.
   When the message "OK." is displayed, copying is completed.

Appendix H Optional Colour Plotter-Printer MZ-1P16



(viewed from the top)



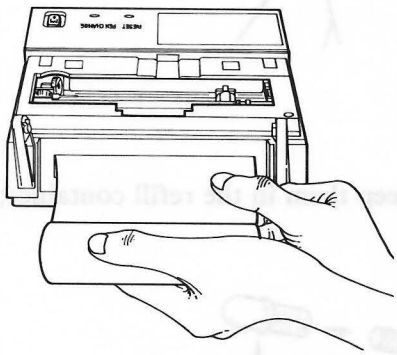
(viewed from the rear)

Note:
A protective sheet is inserted in printer to protect the printer mechanism. Remove the sheet by pressing the paper feed key ( ) before using the printer.

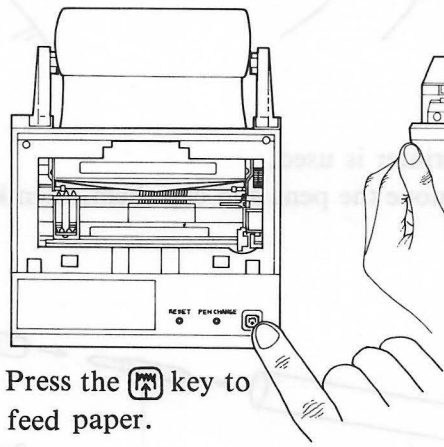
■ Loading roll paper

1. Remove the printer cover.
2. Cut the end of roll paper squarely and insert the paper into the paper inlet. (Do not fold or wrinkle the end of the paper when doing this.)
3. Turn on the MZ-800's power switch and press the (paper feed) key to feed the paper unitl the leading edge exposed 3 to 5 cm above the outlet.
4. Insert the paper shaft into the paper roll and mount it to the paper holders.
5. Refit the printer cover so that the end of paper comes out through the paper cutter.

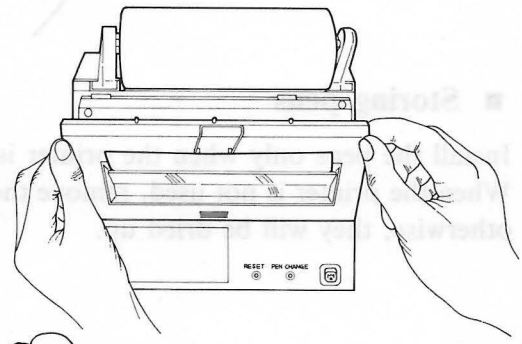
- To remove the roll from the printer for replacement, cut the paper squarely at the paper inlet and press the key.



Insert paper into the paper inlet.



Press the key to feed paper.



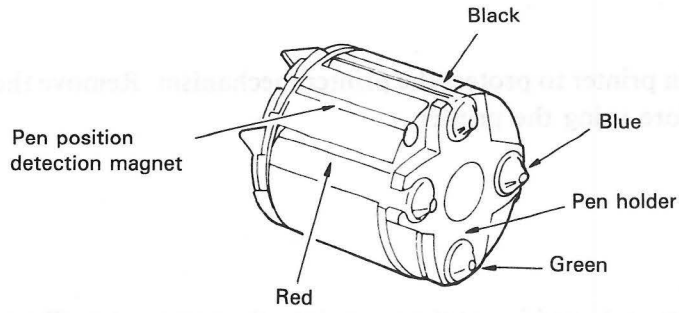
Replace the printer cover.

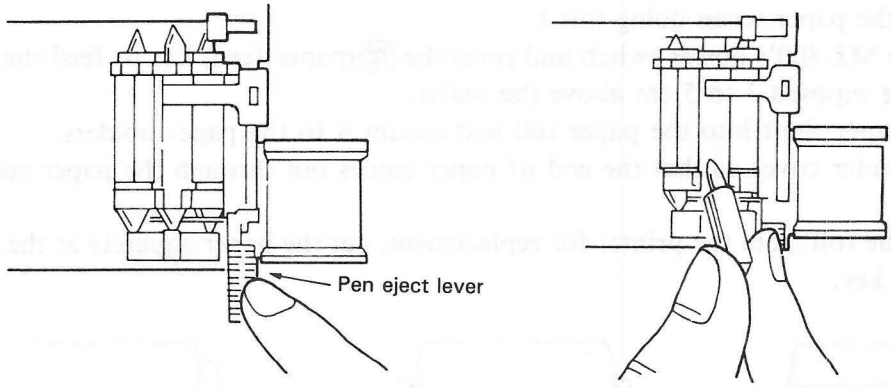
■ Roll paper for the plotter printer is available from your nearest SHARP dealer. Do not use paper other than that specified.

The roll length is 23 to 25 meteres, and the maximum roll which can be loaded is ø 50 mm. The paper will not feed properly if a roll of greater diameter is used, resulting in poor printer quality.

■ Installing/replaceing pens

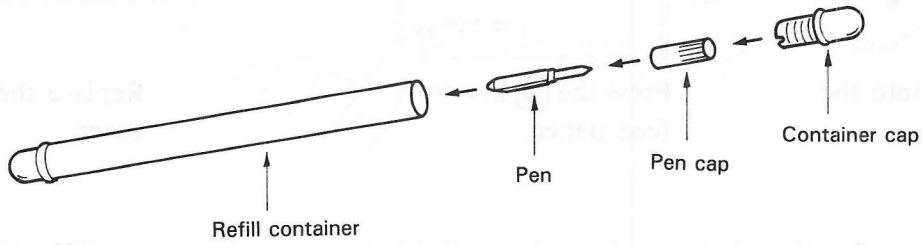
1. Remove the priner cover and press the PEN CHANGE switch with a ball-point pen or similar object; this causes the pen holder to move to the right side of the printer for pen replacement.
2. Press the pen eject lever to eject the pen which is at the top of the holder. When doing this, rest your finger lightly on top of the pen while pushing the eject lever to prevent the pen from falling inside the priter.
3. Insert a new pen.
4. Press the PEN CHANGE switch again to bring another pen to the top of the holder.
5. Replace all four pens (black, blue, green and red) in the same manner. When finished, press the RESET switch.
   Execute the BASIC PTEST command to confirm that all colours are printed correctly.





■ Storing pens

Install the pens only when the printer is used.
When the printer is not used, remove the pens and cap them, then keep them in the refill container; otherwise, they will be dried up.



Note:
Because the ball-point pens use water-soluble ink and the ink may blur if the printed paper becomes wet, the paper should be handled with care.

■ Replacements for the printer pens (ball-point pens) can be purchased at the same dealer you purchased the printer from.

- EA-850B (black: 4 pens)
- EA-850C (black, blue, green, red: 4 pens, 1 of each colour)

■ Self-test

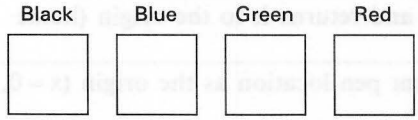
The plotter-printer has the self-test function. Press and hold the [PAPER] (paper feed) key and turn on the MZ-800 power, and the self-test starts. It is recommended to perform the self-test after pens have been replace.

Note:
Be sure to disconnect the interface cable when performing the self-test.

Appendix I Colour Plotter-Printer Control Codes

1 Control codes used in the text mode

- Text code ($01)
  Places the printer in the text mode.
- Graphic code ($02)..... Same as the BASIC PMODE statement.
  Places the printer in the graphics mode.
- Line up ($03)..... Same as the BASIC PSKIP statement.
  Moves the paper one line in the reverse direction. The line counter is decremented by 1.
- Pen test ($04)..... Same as the BASIC PTEST statement.
  Writes the following patterns to start ink flowing from the pens, then sets scale = 1 (40 chr/line), colour = 0.



- Reduction scale ($09) + ($09) + ($09)
  Reduces the scale from 1 to 0 (80 chr/line).
- Reduction cancel ($09) + ($09) + ($0B)
  Enlarges the scale from 0 to 1 (40 chr/line).
- Line counter set ($09) + ($09) + $(ASCII)_2 + (ASCII)_1 + (ASCII)_0$ + ($0D)
  ..... Same as the BASIC PAGE statement.
  Specifies the number of lines per page as indicated by the 3 ASCII bytes code. The maximum number of lines per page is 255. Automatically set to 66 when the power is turned on or the system is reset.
- Line feed ($0A)..... Same as the BASIC PSKIP statement.
  Moves the paper one line in the forward direction. The line counter is incremented by 1.
- Magnify scale ($0B)
  Enlarges the scale from 2 to 1. (26 chr/line)
- Magnify scale ($0C)
  Reduces the scale from 2 to 1.
- Carriage return ($0D)
  Moves the carriage to the left side of the paper.
- Back space ($0E)
  Moves the carriage one column to the left. This code is ignored when the carriage is at the left side of the paper.
- Form feed ($0F)
  Moves the paper to the beginning of the next page and resets the line counter to 0.
- Next colour ($1D)
  Changes the pen to the next colour.

2 Character scale

- The character scale is automatically set to 1 (40 chr/line) when the power is turned on. Afterwards, it can be changed by control codes and commands.
- In the graphics mode, the scale can be changed within the range 0 to 63.
- The scale is set to 1 when the mode is switched from graphics to text.

3 Graphics mode commands

Command type

In the graphics mode, the computer can control the printer with the following commands.
The words in parentheses are BASIC statements which have the same functions as the graphics mode commands.

Command name	Format	Function
LINE TYPE	Lp (p=0 to 15)	Specifies the type of line (solid or dotted) and the dot pitch. p=0 : solid line, p=1 to 15 : dotted line
ALL INITIALIZE	A	Places the printer in the text mode.
HOME (PHONE)	H	Lifts the pen and returns it to the origin (home position).
INITIALIZE (HSET)	I	Sets the current pen location as the origin (x=0, y=0).
DRAW (LINE)	Dx, y, ..., xn, yn ($-999 \leqq x, y \leqq 999$)	Draws lines from the current pen location to coordinates (x_1, y_1), then to coordinates (x_2, y_2), and so forth.
RELATIVE DRAW (RLINE)	JΔx, Δy, ..., Δxn, Δyn ($-999 \leqq \Delta x, \Delta y \leqq 999$)	Draws lines from the current pen location to relative coordinates (Δx_1, Δy_1), then to relative coordinates (Δx_2, Δy_2) and so forth.
MOVE (MOVE)	Mx, y ($-999 \leqq x, y \leqq 999$)	Lifts the pen and moves it to coordinates (x, y).
RELATIVE MOVE (RMOVE)	RΔx, Δy ($-999 \leqq \Delta x, \Delta y \leqq 999$)	Lifts the pen and moves it to coordinates (Δx, Δy).
COLOR CHANGE (PCOLOR)	Cn (n=0 to 3)	Changes the pen colour to n.
SCALE SET	Sn (n=0 to 63)	Specifies the character scale.
ALPHA ROTATE	Qn (n= 0 to 3)	Specifies the direction in which characters are printed.
PRINT	$Pc_1c_2c_3$ ... cn (n=∞)	Prints characters.
AXIS (AXIS)	Xp, q, r (p=0 or 1) (q= $-$999 to 999) (r=1 to 255)	Draws an X axis when p=1 and a Y axis when p=0. q specifies the scale pitch and r specifies the number of scale marks to be drawn.

Command format

There are 5 types of command formats as shown below.

1. Command character only (without parameters)
   A , H , I
2. Command character plus one parameter
   L , C, S , Q
3. Command character plus pairs of parameters
   D , J , M , R
   " , " is used to separater parameters, and a CR code is used to end the parameter list.
4. Command plus character string
   P
   The character string is terminated with a CR code.
5. Command plus three parameters
   X
   " , " is used to separate parameters.

Parameter specification

1. Leading blanks are ignored.
2. Any number preceded by " – " is treated as a negtive number.
3. If the number of digits of a number exceeds 3, only the lower 3 digits are effective.
4. Each parameter is ended with " , " or a CR code. If other than numbers are included in a parameter, subsequent characters are ignored until a comma or CR codde is detected.

Example) D␣␣–135. 21, ...

Ignored

Abbreviated formats

1. Any command can be followed by a one-character command without having to enter a CR code.
   E.g) "HD100, 200" CR is valid and is the same as "H" CR "D100, 200" CR.
2. Any command can be followed by a command with one parameter by separating them with a comma " , ".
   E.g) "L0, S1, Q0, C1, D100, 200" CR is valid.
3. A command with pairs of parameters must be terminated with a CR code.

Data change due to mode switching

The following data changes when the printer is switched from the graphics mode to the text mode.

- X and Y coordiantes
  Y is set to 0 and the origin is placed at the left side of the printable area.
- Direction of characters
  Q is set to 0.
- Character scale
  Character scale is set to 1.
- The line type setting is not affected.

Appendix J Code Tables

■ ASCII code table

MSD is an abbreviation for most significant digit, and represents the upper 4 bits of each code. LSD is an abbreviation for least significant digit, and represents the lower 4 bits of each code. Codes 11_H to 16_H are cursor control codes. For example, executing CALL PRINT (a monitor subroutine) with 15_H loaded into the ACC returns the cursor to the home position. (" H " is not displayed.)

LSD \ MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000		SP	0	@	P					q	n				
1	0001		!	1	A	Q					a					
2	0010		"	2	B	R					e	z	Ü			
3	0011		#	3	C	S						w	m			
4	0100		$	4	D	T					~	s				
5	0101		%	5	E	U						u				
6	0110		&	6	F	V				t	i					
7	0111		'	7	G	W				g		o				
8	1000		(	8	H	X				h	Ö	l				
9	0001		)	9	I	Y					k	Ä				
A	1010		*	:	J	Z				b	f	ö				
B	1011		+	;	K			°	^	x	v	ä				£
C	1100		,	<	L					d						
D	1101	CR	—	=	M					r	ü	y				
E	1110		.	>	N					p	ß					
F	1111		/	?	O					c	j					π

■ ASCII code table

When using the colour plotter-printer, graphics characters other than those shown above cannot be printed, but the curresponding hexadecimal code is printed in a different pen colour.

MSD / LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P			}		q	n				
1				1	A	Q					a					
2			"	2	B	R				e	z					
3			#	3	C	S				`	w	m				
4			$	4	D	T				~	s					
5			%	5	E	U					u					
6			&	6	F	V				t			→			
7			'	7	G	W				g		o		▭		
8			(	8	H	X				h		!				
9			)	9	I	Y					k					
A			*	:	J					b	f					
B			+	;	K	[		▫	^	x	v					£
C			,	<	L	\				d						↓
D			−	=	M	]				r		y				
E			.	>	N	↑				p	B	{				
F			/	?	O	←				c				▭		π

Appendix K Error Messages Generated by the Monitor

[Cassette]

CMT: Loading error
An error occurred during data loading.

CHECK SUM ER.
An error was detected in the check sum for the loaded file.

Make ready CMT:
An attempt was made to access data before the PLAY button was pressed, or when no cassette was inserted in the data recorder.

[MZ disk]

QD: Loading error
An error occurred during data loading.

QD: File mode error
The type of the starting file on the disk set in the drive at the power-on sequence was not OBJ.

QD: File not found
The specified OBJ file was not found.

QD: Hard err
A hardware error occurred.

Already exist err
A filename which is the same as that specified for the S command had been already cataloged on the disk.

QD: Write protect
An attempt was made to access a write-protected disk.

QD: Not ready
An attempt was made to access data for the disk when the disk holder was opened or when no disk was inserted in the drive.

QD: No file space err
Insufficient free space was left when a file was saved by the S command.

QD: Unformat err
The disk to be accessed was unformatted.

QD: Bad disk err
The disk to be accessed was defective.

Make ready QD
An attempt was made to access data when the disk holder was opened or when no disk was set in the drive.

[Floppy Disk]

FD: Loading error
An error occurred during data loading.

FD: Not master
The disk set in drive-1 was not master disk.

Make ready FD:
An attempt was made to access data when the lever was not locked or when no disk was set in the drive.

Appendix L Error Messages Generated by BASIC

When an error occurs during BASIC operations, either of the following error messages (1) and (2) is displayed on the screen.

(1) (Type of error) error
(2) (Type of error) error in (Line number)

Message (1) is generated when a command is entered from the keyboard,while message (2) is generated during program execution.

Error No.	Message Displayed	Description
1	Syntax error	A statement does not conform to the syntax rules of BASIC.
2	Overflow error	The magnitude of a numeric value exceeds the limits.
3	Illegal data error	A numeric value or variable which does not conform to the numeric rules of BASIC was encountered.
4	Type mismatch error	The types of data and variable do not match.
5	String length error	The number of characters included in a string exceeds 255.
6	Memory capacity error	Insufficient memory space is available for the processing required.
7	Array def. error	An array was to be expanded or undefined array name was specified.
8	Line-length error	The length of a line exceeds the limits.
10	GOSUB nesting error	The number of nested GOSUB statements exceeds 15.
11	FOR-NEXT error	The number of nested FOR-NEXT statements exceeds 15.
12	DEF FN nesting error	The number of nested function definitions using the DEF FN statement exceeds 6.
13	NEXT error	A NEXT statement was encountered without a corresponding FOR statement.
14	RETURN error	A RETURN statement was encountered without a corresponding GOSUB statement.
15	Un def. Function error	A call was made to an undefined function.
16	Un def. line error	A non-existent line number or label was specified.
17	Can't CONT error	Program continuation with a CONT statement is impossible.
18	Memory protection error	An attempt was made to write data in the area reserved for the BASIC interpreter.

Error No.	Message Displayed	Description
19	Instruction error	A direct command was included in the program or an indirect statement was used in the direct mode.
20	Can't RESUME error	A RESUME statement cannot be used.
21	RESUME error	An attempt was made to execute a RESUME statement even though no error occurred.
22	PAL error	Palette block number is out of range.
24	READ error	A READ statement was encountered without a corresponding DATA statement.
29	Framing error	Framing error
30	Overrun error	Overrun error
31	Parity error	Parity error
40	File not found error	An attempt was made to access a non-existent file.
42	Already exist error	An attempt was made to save a file under a filename which already existed.
43	Already open error	An attempt was made to open a file already opened.
44	Not open error	An attempt was made to access, CLOSE, or KILL a file without opening it.
46	Write protect error	An attempt was made to write data to a write-protected file.
51	Too many files error	An attempt was made to store more than 32 files in the RAM file board.
53	No file space error	Free space for storing files is insufficient.
58	Dev. name error	An invalid device name was specified.
59	Can't execute error	An attempt was made to execute a statement for an invalid device.
60	Illegal filename error	An illegal filename was specified.
61	Illegal filemode error	A file was accessed in an illegal mode.
63	Out of file error	An end of file was encountered during a read operation of the cassette.
64	Logical number error	An error was detected in the logical number.
65	LPT: not ready error	The printer is not connected or not on-line, or a malfunction has occurred in the printer mechanism.
68	Dev. mode error	An error was detected in the device mode.
69	Unprintable error	An error occurred which does not have a message.
70	Check sum error	An error was detected in check sum data.

Appendix M Index

A
ABS 5-10
AND 5-8
ASC 5-12
ATN 5-10
AUTO 6-3
AXIS 6-81

B
BLINE 6-61
BOOT 6-42
BOX 6-61
BYE 6-42

C
CHAIN 6-49
CHR$ 5-13
CIRCLE 6-62
CLOSE # 6-54
CLR 6-6
CLS 6-9
COLOR 6-56
CONSOLE 6-9
CONT 6-7
COS 5-10
CSRH 5-5
CSRV 5-5
CURSOR 6-10

D
DEF FN 6-34
DEF KEY 6-36
DEFAULT 6-55
DELETE 6-4,48
DIM 6-30
DIR 6-43

E
END 6-13
EOF (#) 6-53
ERL 5-5
ERN 5-5
EXP 5-10

F
FOR ~ NEXT 6-14

G
GET 6-29
GOSUB ~ RETURN 6-18
GOTO 6-16
GPRINT 6-80

H
HCOPY 6-83
HSET 6-80

I
IF ~ THEN ~ :ELSE 6-20
IF ~ GOSUB 6-23
IF ~ GOTO 6-22
INIT 6-37
INP@ 6-84
INPUT 6-29
INPUT # 6-53
INT 5-10

K
KEY LIST 6-36
KILL # 6-54

L
LABEL 6-16
LEFT$ 5-11
LEN 5-13
LET 6-11
LIMIT 6-86
LINE 6-60
LIST 6-4
LIST/P 6-83
LN 5-11
LOAD 6-44
LOG 5-11

M
MERGE 6-50
MID$ 5-11
MUSIC 6-68

N
NEW 6-6
NEW ON 6-6
NOISE 6-72
NOT 5-8

O
ON ERROR GOTO 6-87
ON ~ GOSUB 6-19
ON ~ GOTO 6-17
OR 5-8
OUT@ 6-85

P
PAGE 6-75
PAI 5-11
PAINT 6-63
PAL 6-57
PATTERN 6-64
PCIRCLE 6-82
PCOLOR 6-75
PEEK 6-84
PHOME 6-79
PLINE 6-77
PLOT 6-83
PMODE 6-73
PMOVE 6-78
POINT 6-67
POKE 6-84
POSH 5-5
POSITION 6-65
POSV 5-5
PRINT 6-24
PRINT # 6-51
PRINT/P 6-76
PRINT/P USING 6-76
PRINT USING 6-25
PSKIP 6-75
PTEST 6-73

R
RAD 5-11
READ ~ DATA 6-31
REM 6-11
RENAME 6-48
RENUM 6-5
RESET 6-59
RESTORE 6-33
RESUME 6-88
RIGHT$ 5-11
RLINE 6-78
RMOVE 6-79
RND 5-13
ROPEN # 6-52
RUN 6-8,43

S
SAVE 6-46
SEARCH 6-5
SET 6-58
SIN 5-10
SIZE 5-5
SGN 5-10
SOUND 6-71
SPC 5-12
SQR 5-10
STICK 5-14
STOP 6-12
STR$ 5-12
STRIG 5-14
SYMBOL 6-66

T
TAB 5-11
TAN 5-10
TEMPO 6-71
TI$ 5-5
TROFF 6-35
TRON 6-35

U
USR 6-85

V
VAL 5-12
VERIFY 6-47

W
WAIT 6-42
WOPEN # 6-51

X
XOR 5-8

Appendix N Specifications

CPU	Z80A-CPU
Clock	3.5469 MHz
Memory	ROM 16K bytes RAM 64K bytes VRAM 16K bytes Can be expanded to 32 K bytes. (option)
Display	I/F : RF, Video, RGB Graphic display : 320 × 200 dots 640 × 200 dots
Cassette	Standard audio cassette tape Data transfer speed ; 1200 bits/sec. Data transfer system ; SHARP PWM
Key layout	ASCII standard main keyboard Special function keys Cursor control keys Cassette tape deck control keys
Editing function	Cursor control; up, down, left, right, home, clear Deletion, insertion
Clock function	Built-in
Power supply	Local supply rating voltage
Temperature	Operating temp; 10° to 35°C
Humidity	20% ~ 80% (no condensation)
Weight	MZ-811; 4.0 kg MZ-821; 4.3 kg
Dimensions	Width : 440 mm Depth : 305 mm Height: 109 mm
Accessories	Power cable Owners manual Cassette Monitor cable Definable key lavel Graphic key lavel

This apparatus complies with requirements of BS 800 and EEC directive 82/499/EEC.

Dieses Gerät stimmt mit den Bedingungen der EG-Richtlinien 82/499/EWG überein.

Cet appareil répond aux spécifications de la directive CCE 82/499/CCE.

Dit apparaat voldoet aan de vereiste EEG-reglementen 82/499/EEG.

Apparatet opfylder kravene i EF direktivet 82/499/EF.

Questo apparecchio è stato prodotto in conformità alle direttive CEE 82/499/CEE.

Printed in Japan
Gedruckt in Japan
Imprimé au Japon
Stampato in Giappone

© 1984 SHARP CORPORATION

4L 5.6-I(TINSE1295ACZZ)②

SHARP CORPORATION
OSAKA, JAPAN