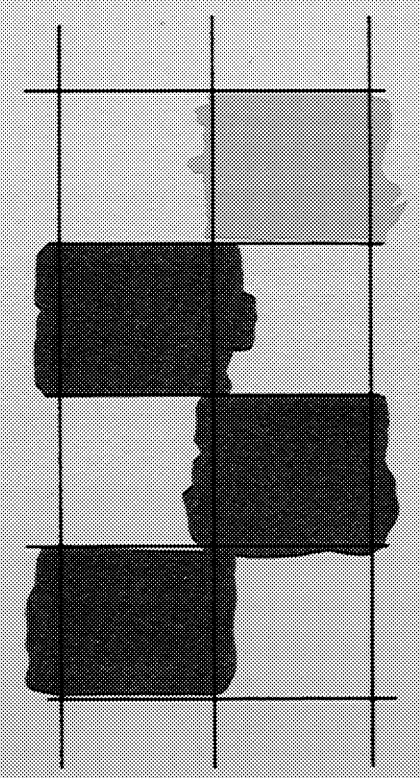
TOSHIBA

PROGRAMMABLE CONTROLLER

ASCII / BASIC MODULE



EX250/500
EX2000

USER'S MANUAL
INSTALLATION AND SPECIFICATIONS

Important information

Misuse of this equipment can result in property damage or human injury. Because controlled system applications vary widely, you should satisfy yourself as to the acceptability of this equipment for your intended purpose. In no event will Toshiba Corporation be responsible or liable for either indirect or consequential damage or injury that may result from the use of this equipment. No patent liability is assumed by Toshiba Corporation with respect to the use of information, illustrations, circuits, equipment, or examples of applications in this publication.

The Toshiba Corporation reserves the right to make changes and improvements to this publication and/or related products at any time without notice. No obligation shall be incurred, except as noted in this publication.

ASCII/BASIC MODULE

OPERATION MANUAL

Important Information

Misuse of this equipment can result in property damage or human injury. Because controlled system applications vary widely, you should satisfy yourself as to the aceptability of this equipment for your intended purpose. In no event will Toshiba Corporation be responsible or liable for either indirect or consequential damage or injury that may result from the use of this equipment.

No patent liability is assumed by Toshiba Corporation with respect to use of information, illustrations, circuits, equipment, or examples of aplication in this publication.

Toshiba Corporation reserves the right to make changes and improvements to this publication and/or related products at any time without notice. No obligation shall be incurred other than as noted in this publication.

Contents

Contents

1. How to Use This Manual

This chapter describes the construction of the manual, introduces the operation manuals of related equipment, explains special marks used in the text, and defines phrases and terms used in this document. Be sure to familiarize yourself with the information presented in this chapter before reading the rest of this ASCII/BASIC Module Operation Manual.

1.1 Inside This Manual

This manual has been prepared to inform you about the installation and operation of the ASCII/BASIC module.

This ASCII/BASIC Module Operation Manual consists of six chapters and six appendices. For those users who will be operating the ASCII/BASIC module for the first time, the manual proceeds from a general description of the ASCII/BASIC module to installation, basic operation, programming, and programming application. The appendices provide information about and descriptions of the specifications of the ASCII/BASIC module and of the BASIC language used by this module. The descriptions in this manual assume that you have a basic knowledge of the programmable controller and the BASIC language. If you need to know more about the programmable controller, you are requested to read the operation manuals listed in section 1.2. For more information about the BASIC language, read the operation manuals published by the manufacturers of personal computers sold in the market.

1.2 Related Documents

The ASCII/BASIC module is one of the I/O modules used by the EX250/500/2000 programmable controllers. Knowledge of the basic operations of these programmable controllers is essential before operating the ASCII/BASIC module. Related operation manuals are:

EX250/500 USER'S MANUAL OPERATION
(UM-EX250U*-E203)
EX250/500 USER'S MANUAL WIRING AND MAINTANANCE
(UM-EX25U-E003)
EX2000 USER'S MANUAL INSTALLATION AND SPECIFICATION
(UM-EX2K***-E001)

1.3 Symbols Used in This Manual

This manual contains information marked as follows:

You should pay special attention to information preceded by the following designations.

HINT

"Hints" provide helpjul suggestions for most effective programming and /or operation of the equipment.

NOTE "Notes" call the reader's attention to information considered important for full understanding of programming procedures and/or operation of the equipment.

CAUTION "Cautions" call the reader's attention to conditions or pracitices that could damage equipment or render it temporarily inoperative.

1.4 Terminology

Many terms particular to PC operation are defined in the glossary of "Guidebook to Manuals." You should familiarize yourself with these terms before proceeding. Additional terms used to explain programming are provided in a glossary in the appendix of this maunal. The following is a list of abbreviations and acronyms used.

ACO	AC output
A/D	analog/digital
AI	analog input
AO	analog output
AWG	American wire gage
BCD	binary coded decimal
CMOS	complementary metal oxide semiconductor
CPU	central processing unit
CSA	Canadian Standards Association
DI	DC input
DO	DC output
GP100	graphic programmer
GP100AP	graphic programmer, stand-alone model
GND	ground
H	hexadecimal (when it appears in front of alphaunmeric string)
Hz	Hertz
INP	AC input
I/O	input/output
LCD	liquid crystal display
LED	light-emitting diode
LSB	least significant bit
LSI	large-scale integration
MP100	miniprogrammer
MSB	most significant bit
NC	normally closed
NEMA	National Electrical Manufacturers Association
NO	normally open
PC	programmable controller
PI	pulse input
PR100	PROM writer

PROM	programmable read-only memory
RAM	random access memory
RO	relay output
RS-232C, RS-422	serial interfaces
RTD	resistance temperature detector (input)
UL	Underwriters' Laboratories, Inc.
Vac	ac voltage
Vdc	dc voltage

Especially, the following terms are used in this manual:

- ASCII/BASIC module : ASCII module
- EX250, EX500 and EX2000 CPUs (generically) : EX CPUs
- MP100 : MP
- HP100 : HP
- MCS® BASIC-52 (BASIC interpreter) : BASIC-52
- GP100, GP100AP, GP110, GP110AP1, GP110AP2 (generically) : GP

The terms and notation methods used in connection with the BASIC language are described in "Appendix 5. Description of BASIC-52 Language." In chapter 4, the following notation method is used:

- [CR] shows input of the Carriage Return [ENTER] key on the console device keyboard.

MCS® BASIC-52 is a registered trademark of Intel Corporation, of the United States.

2.1 Outline of the Module

The ASCII/BASIC module (herein after called "ASCII module")has two RS-232C ports. Through these ports, peripheral devices, such as a bar code reader, printer, and display, and the EX250/500/2000 are connected for exchange of data. The ASCII module can also process complex numerical operations that cannot be performed by the EX CPU using the BASIC programs instead of the EX CPU.

This is processed entirely by executing the BASIC programs created by the user. Up to 255 BASIC programs can be saved in the EEPROM of the ASCII module. The basic and expansion types have memory capacities of 32 k and 64 k bytes, respectively.

Figure 2.1 shows a sample system configuration connecting a console device and printer to the ASCII module.

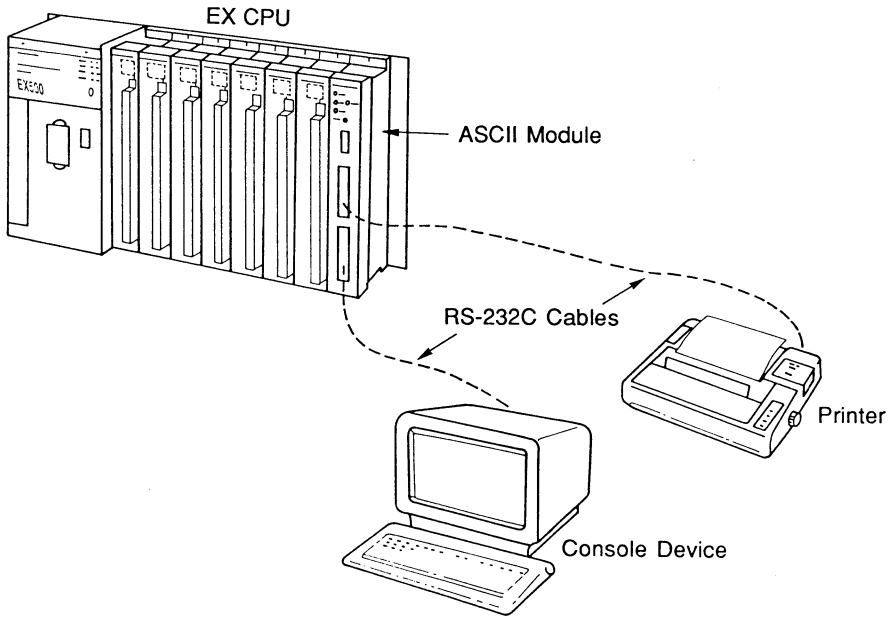


Figure 2.1 Sample System Configuration

2.2 Module Configuration

Figure 2.2 shows the front panel of the ASCII module with the names of the components.

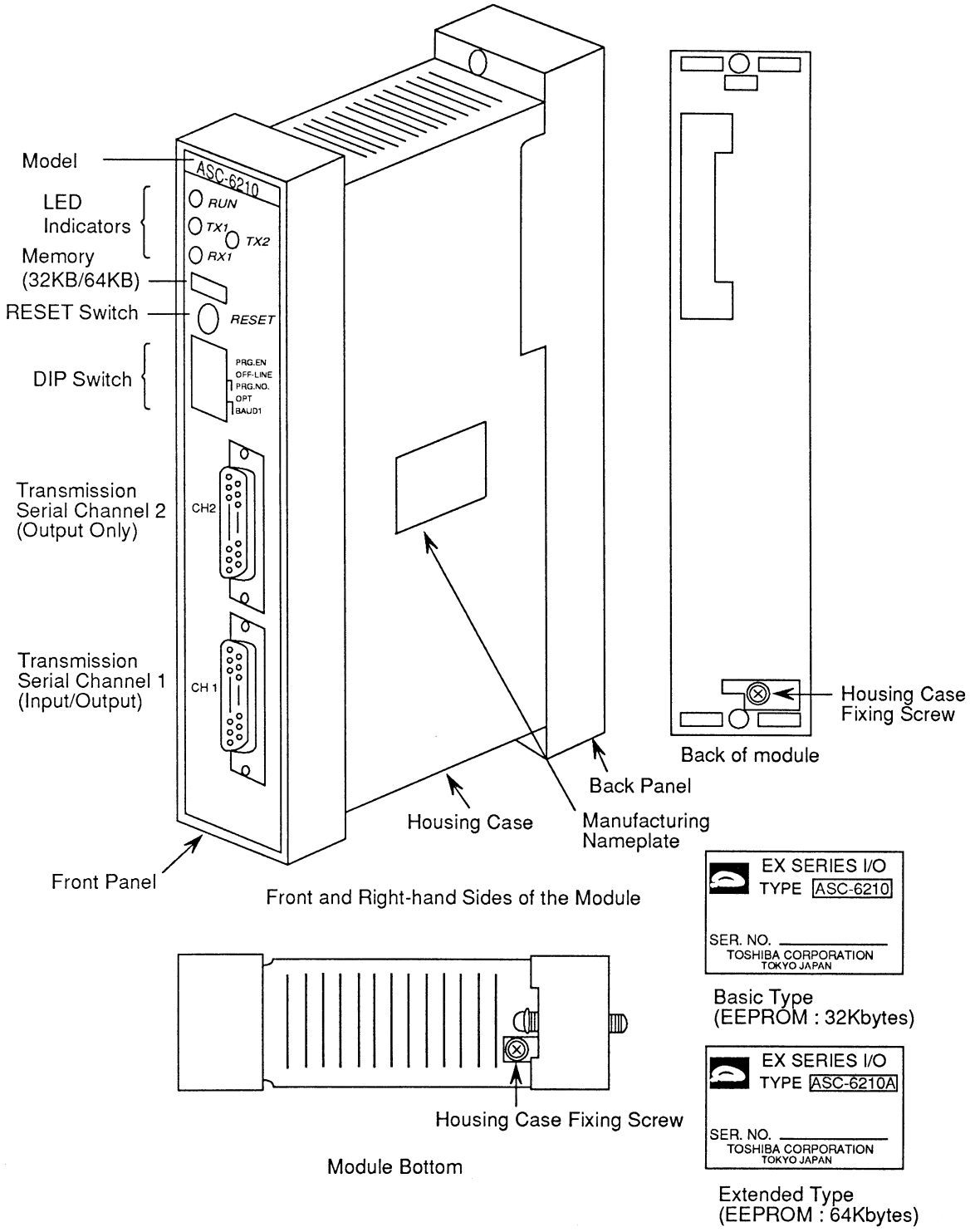


Figure 2.2 Front Panel of ASCII Module

The functions of the LED indicators and switches are summarized below.

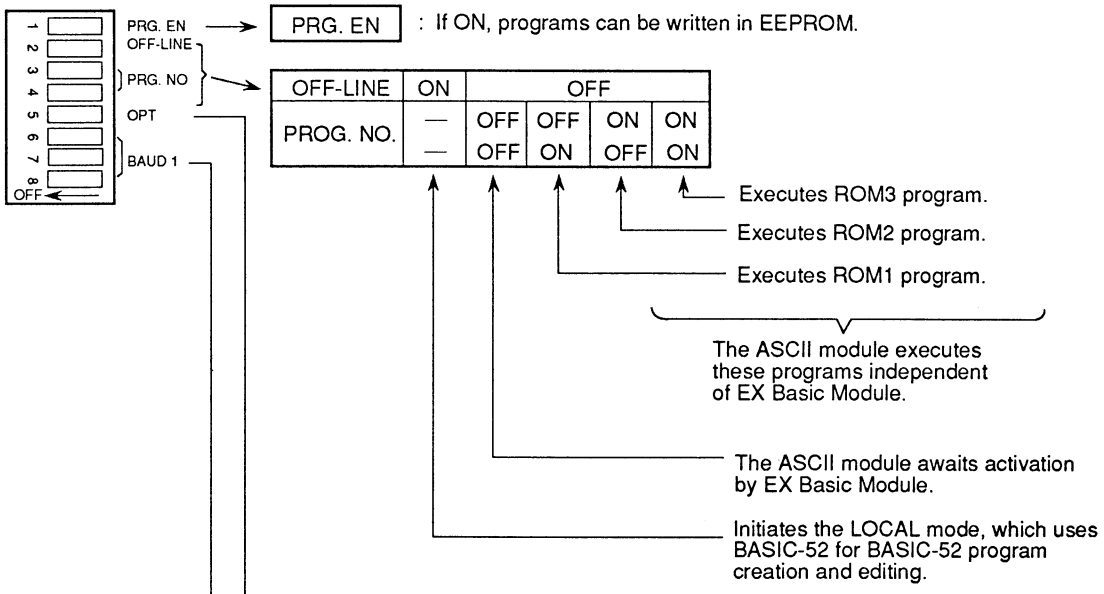
LED Indicators

RUN	Lights when the ASCII module is operating normally.
TX1	Lights if there is transmission data from Serial Channel 1.
TX2	Lights if there is transmission data from Serial Channel 2.
RX1	Lights if there is reception data to Serial Channel 1.

Internal CPU Reset Switch

Press this switch after changing setting of the DIP switch. It resets the internal CPU of the ASCII module. Press this switch also if the ASCII module functions improperly.

DIP Switch



BAUD1	OFF OFF OFF	OFF OFF ON	OFF ON OFF	OFF ON ON	ON OFF OFF	ON OFF ON	ON ON OFF	ON ON ON
CH1 baud rates	110	300	600	1200	2400	4800	9600	19200

Set the CH1 baud rates as shown above.

2.3 Module Functions

The ASCII module uses a micro controller called 8052 AH-BASIC as the CPU. This CPU contains a BASIC interpreter called MCS® BASIC-52 ((herein after called "BASIC-52"). Thus, the ASCII module can process complex numerical operations, which cannot be processed by ladder sequence commands of the EX CPU, using the BASIC-52 programs on behalf of the EX CPUs.

Through the two RS-232C ports of the ASCII module, peripheral devices such as a bar code reader, printer and display can be connected to the EX CPUs for the exchange of data. Figure 2.3 shows the flow of data.

Two interface types are used between the EX CPUs and ASCII module. One method is a combination of the usual ladder sequence commands. The other method uses two special commands: READ and WRITE. (These two commands are grouped and are called the READ/WRITE commands below.)

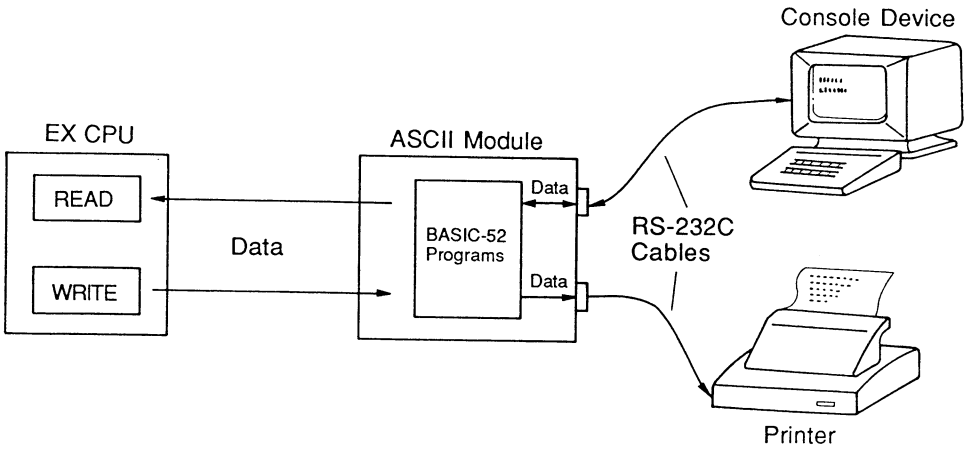


Figure 2.3 Flow of Data Among EX CPU, ASCII Module, and Peripheral Devices

Regardless of which method is used, the EX CPU first designates the number of the BASIC-52 program to be executed by the ASCII module. As the ASCII module executes this program, data is exchanged among the EX CPU, ASCII module and peripheral devices.

By executing the READ command, the EX CPU can assign the ASCII module a function, such as to fetch data from the peripheral devices, so that the data set by the ASCII module can be received and stored by the EX CPU.

By executing the WRITE command, data set by the EX CPU is transmitted to the ASCII module so that the data can be output from the ASCII module to the peripheral devices.

When using the READ/WRITE commands during interface between the EX CPU and ASCII module, a maximum of 500 words of data with EX2000, or 250 words of data with EX250/500, can be exchanged. In this case, the ASCII module acts as an option module.

An EX CPU that does not have the READ/WRITE command functions can interface with the ASCII module by using the table transfer command or other command during the ladder sequence. (To distinguish it from the READ/WRITE commands, the ladder sequence program is called the READ/WRITE sequence.) In this case, the ASCII module acts as an I/O module of the X+Y04W type.

During the execution of an ASCII Module BASIC-52 command, data type conversion routines and flag processing routines specially provided for data transfer can be used. The user can easily exchange data between the EX CPU and ASCII module at high speed by calling these routines during the execution of a BASIC-52 program. This gives a variation to the interface between the EX CPU and ASCII module. There are five types of interfacing methods:

(1) READ processing

This processing hands data over from the ASCII module to the EX CPU.

(2) WRITE processing

This processing hands data over from the EX CPU to the ASCII module.

(3) WRITE with ANSWER processing

This processing hands data over from the EX CPU to the ASCII module and returns the result of processing performed by the ASCII module based on this data to the EX CPU as an "answer."

(4) Continuous READ processing

Processing of handling data over from the ASCII module to the EX CPU is performed continuously.

(5) READ/WRITE simultaneous execution processing

READ and WRITE processing can be performed simultaneously within one BASIC program on the ASCII module side.

Refer to CHAPTER 6. PROGRAMMING APPLICATIONS for the details of each mode.

This chapter describes the installation and mounting of the ASCII module, the proper switch settings, and the power-on and start-up methods.

3.1 Installation Precautions

The ASCII module installation precautions can be grouped into environmental and mounting precautions.

3.1.1 Installation environment

DO NOT install the ASCII module in a location subject to any of the following conditions:

(1) The temperature falls below 0°C (32°F) or goes above 55°C(131°F).

(2) The relative humidity is less than 20% or more than 90%.

(3) Rapid temperature changes cause condensation.

(4) Vibration exceeds the allowable value.

(5) Shocks exceed the allowable value.

(6) Corrosive or combustible gas exists.

(7) Places that are dusty or salty or contain much iron.

(8) Exposed to direct sunlight.

Pay special attention to the following items when installing the frame containing the module.

(1) Securely ground housing frames if there is any high-frequency equipment.

(2) Check that no current is leaking from other frames or equipment if the channel base is shared with other frames.

(3) Separately install the RS-232C transmission cable connected to the ASCII module, keeping it at a distance from other wires connected to strong electric equipment. (Never tie them together.)

3.1.2 Mounting precautions

The ASCII module can be mounted freely in a basic or expansion unit. However, because transmission signals for it are sent on a weak electric system, the wires must not be tied with other external wires of input/output modules using a strong electric system. This is necessary to ensure maximum noise resistance. Figure 3.1 Shows basic layout and wiring considerations.

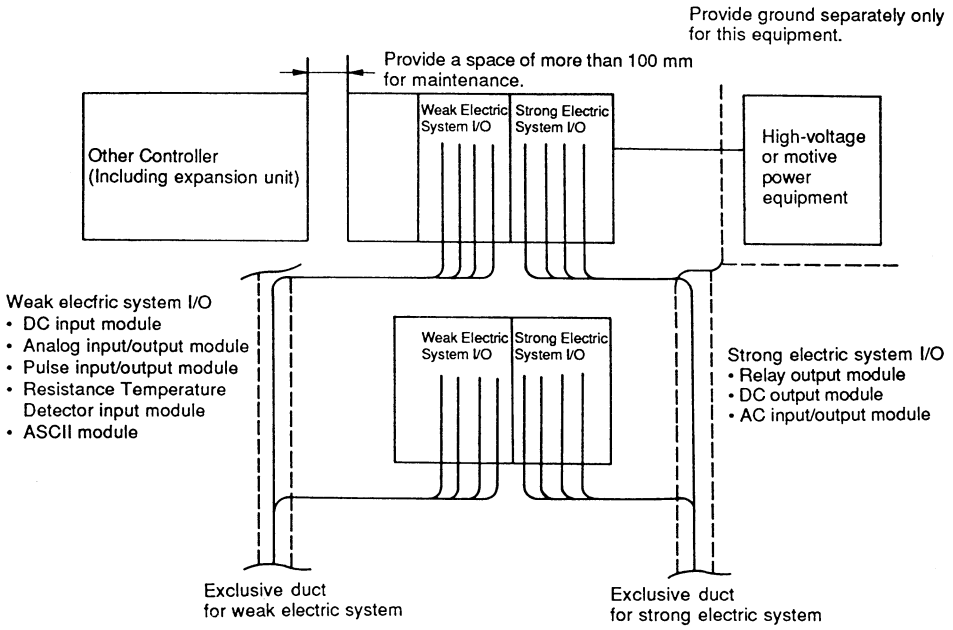


Figure 3.1 Layout and Input/Output Wiring

Figure 3.1 Layout and Input/Output Wiring

CAUTION

1. Do not put ducts for weak and strong electric wires and cables in contact each other.
2. Do not put the RS-232C transmission cable in the wiring duct. Do not install it near strong electric cables.
3. Do not install strong electric cables near the module.

3.2 Switch Settings

Jumper and DIP switch setting should be performed before mounting the ASCII module.

3.2.1 Jumper setting

It is necessary to change the type of the ASCII module depending on the version of the EX CPU being used.

Set the ASCII module as an option module if used with an EX CPU that can use the READ/WRITE commands.

Set the ASCII module as an I/O module of the X+Y04W type if used with an EX CPU that does not have the READ/WRITE commands.

The versions of the EX CPUs that can use the ASCII module as an option module (that is, the READ/WRITE commands can be used) and of GP are shown below.

Model Name	Version that can use READ/ WRITE commands	Version Notation
EX 250	Ver. 2. 01 and after	EX-V 2.01
EX 500	Ver. 2. 01 and after	EX-V 2.01
EX 2000	Ver. 2.1 and after	EX 2000 V 2.1
GP 110	Ver. 1.0 and after	GP 110-V 1.0
GP 110 AP 1		GPAP 1-V 1.0
GP 110 AP 2		GPAP 2-V 1.0

The jumper pin inside the module determines the ASCII module type. This jumper is set during preshipment adjustment at the factory so that the ASCII module will function as an option module. This jumper setting must be changed if the module is to be used with an EX CPU that does not have the READ/WRITE command functions.

To change the jumper setting:

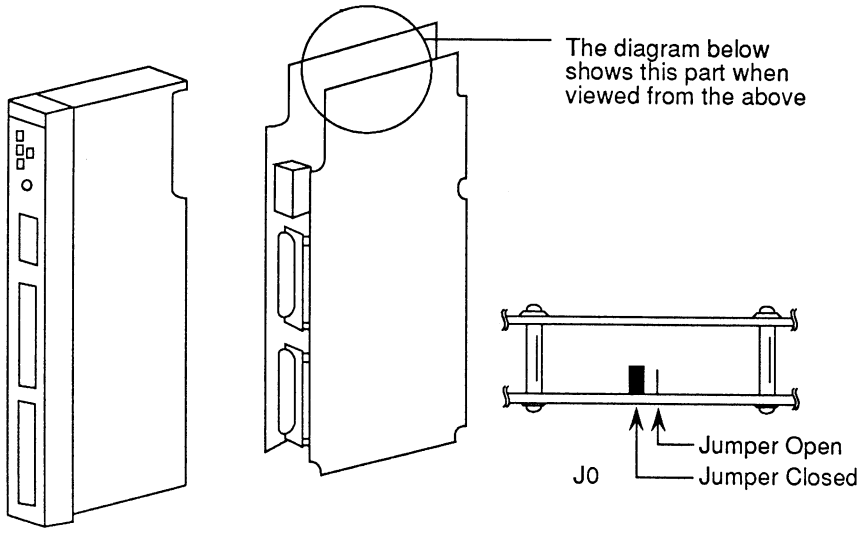
(1) Remove the housing case fixing screws on the bottom and back of the module.

(2) Remove the back panel from the housing case.

CAUTION ▼△▼ A pin is provided to fit a slot in the housing case in the upper part of the back panel. To remove the back panel from the housing case, remove the lower half of the panel, then remove the upper half of the back panel from the housing case while disengaging the pin from the slot.

(3) Remove the board from the housing case.

(4) Check if the jumper is set in the position shown in the diagram.



(5) The ASCII/BASIC module status will change as shown below by jumper setting.

ASCII/BASIC Module Status	Jumper Setting
READ/WRITE command	o
Conventional ladder sequence	x

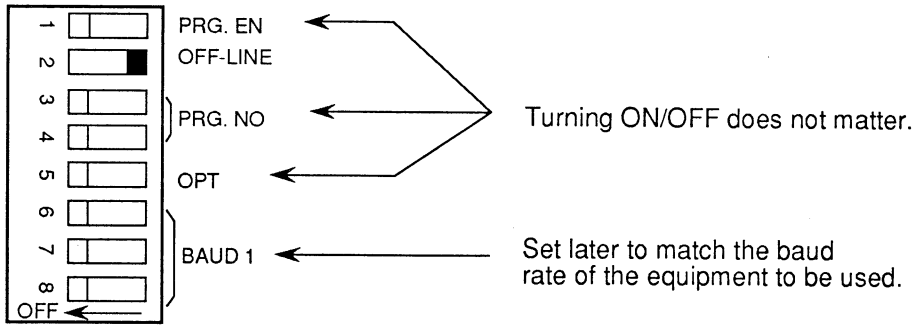
o: Closed

x: Open

(6) Put the module back in the housing case and fix the module by the housing case fixing screws.

3.2.2 DIP switch setting

Set the ASCII module to the LOCAL mode after turning ON the power so that the BASIC programs can be created and edited. Switch on the switch second from the top in the DIP switch installed on the front panel, as shown below.



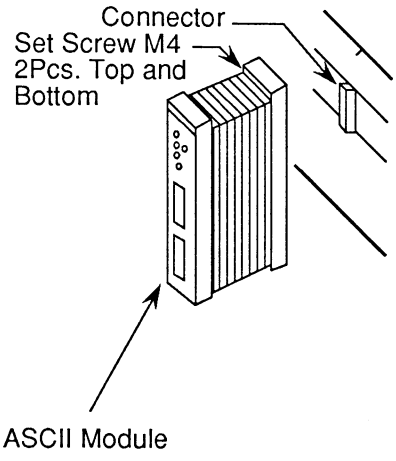
3.3 Mounting Method

Mount the ASCII module as follows, paying strict attention to the precautions mentioned in section 3.1.

(1) Check that the connector interiors and ASCII module connectors are clean before mounting the module. Clean them with alcohol or other solution if they are dirty.

(2) Make sure that power is off when mounting or removing the module.

CAUTION Removing or mounting the module while power is on may cause a module failure.

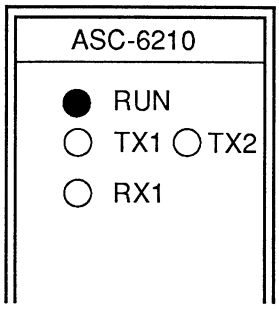


(3) Connect the ASCII module to the base unit connector.

(4) Fix the module using the two top and bottom mounting screws (M4).

3.4 Power On

Turn on power to the EX CPU. The ASCII module is operating properly if the four LED lamps on the front panel of the module are as shown below:



● : LIT

○ : OFF

If the LED lamps are not as shown above, take action in accordance with part 4.1 of "APPENDIX 4. ERROR MESSAGE AND TROUBLESHOOTING LISTS."

Utilizing a simple example, operation of the ASCII module is described in this chapter. Sections 4.1 and 4.2 cover points that should be understood before operating the ASCII module.

4.1 BASIC-52 Programs

This section provides a general description of the BASIC-52 programs used by the ASCII module.

The interface between EX250/500/2000 and ASCII peripheral equipment using the ASCII module requires prior creation of the BASIC-52 programs, not only on the EX CPU side but also on the ASCII module side.

Connect a personal computer or other device that has an RS-232C port to CH1 of the module to create the BASIC-52 programs in the ASCII module.

Save created programs in the EEPROM of the ASCII module.

The BASIC-52 programs are saved in the EEPROM in order by assigned program numbers, as shown in Figure 4.1. These numbers become parameters to load the desired programs selected from the programs saved in the EEPROM.

A maximum of 255 programs can be saved.

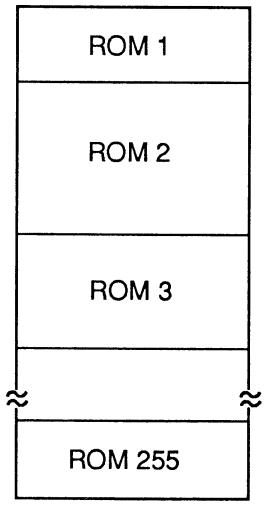


Figure 4.1 Method of Saving Programs in ASCII Module

4.2 Programming Procedures

This section describes the general procedures for incorporating the ASCII module into the system and operating it when there are no BASIC-52 programs in the ASCII module.

Perform all required steps described in CHAPTER 3 and check the LED lamps to ensure that if the ASCII module has been started up properly.
Then:

Step 1: Connect a console device, such as a personal computer, to the ASCII module and create a BASIC-52 program.

Step 2: Make I/O allocation through ,the EX CPU, and create a ladder sequence program.

Step 3: Connect the EX CPU, ASCII module, and peripheral devices. Reset the switches of the ASCII module.

Step 4: Execute the ladder sequence program in the EX CPU.

After this, the ASCII module executes the process routines of the program number designated during the ladder sequence program and of the BASIC-52 program in the EEPROM corresponding to this number:

- Outputting data to the display or printer connected to the ASCII module.
- Storing data input by a peripheral device, such as a bar code reader, in the area designated in the ladder program in the EX CPU.

4.3 Execution of a Sample Program

We'll use a simple example to examine the operation of the ASCII module.

The process proceeds roughly as follows.

(1) The EX CPU sends (WRITEs) five pieces of data to the ASCII module (actual data: 4 pieces; number of words: 1 piece).

(2) The ASCII module calculates square and cube of the received data, and it displays the result of the calculation on the console.

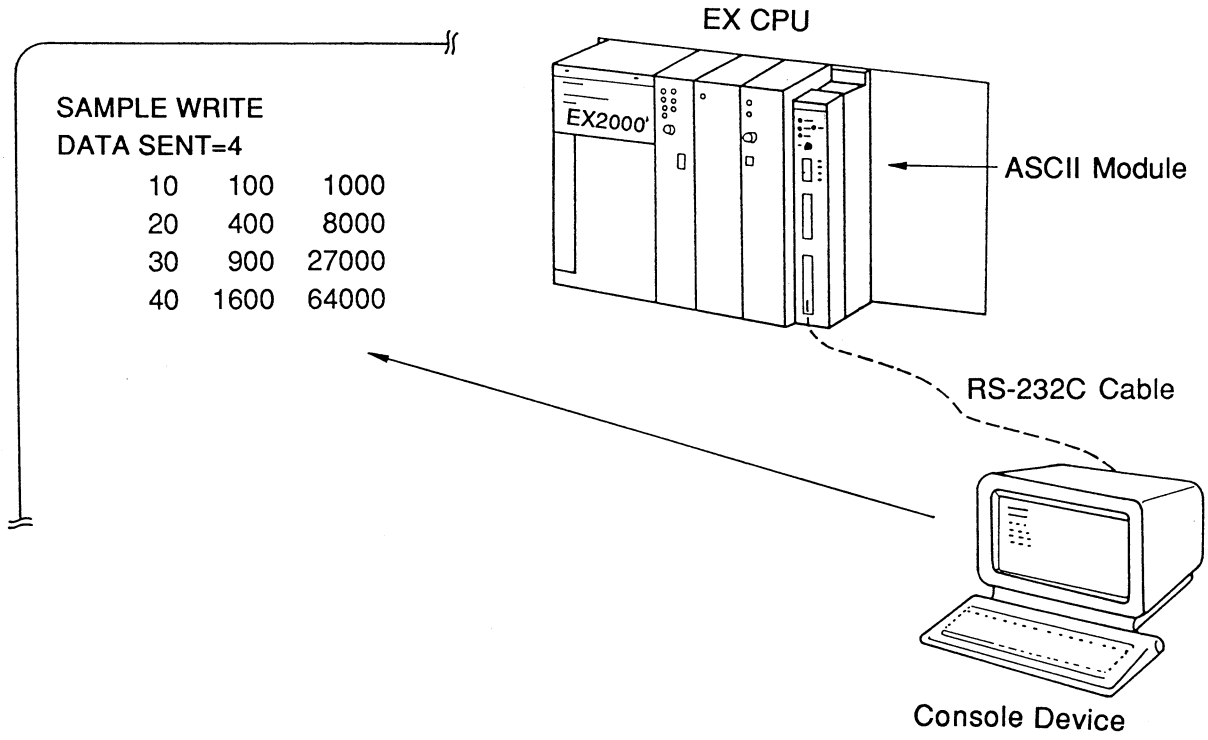


Figure 4.2 Results of Sample Program Execution

Preparations: Install the following items

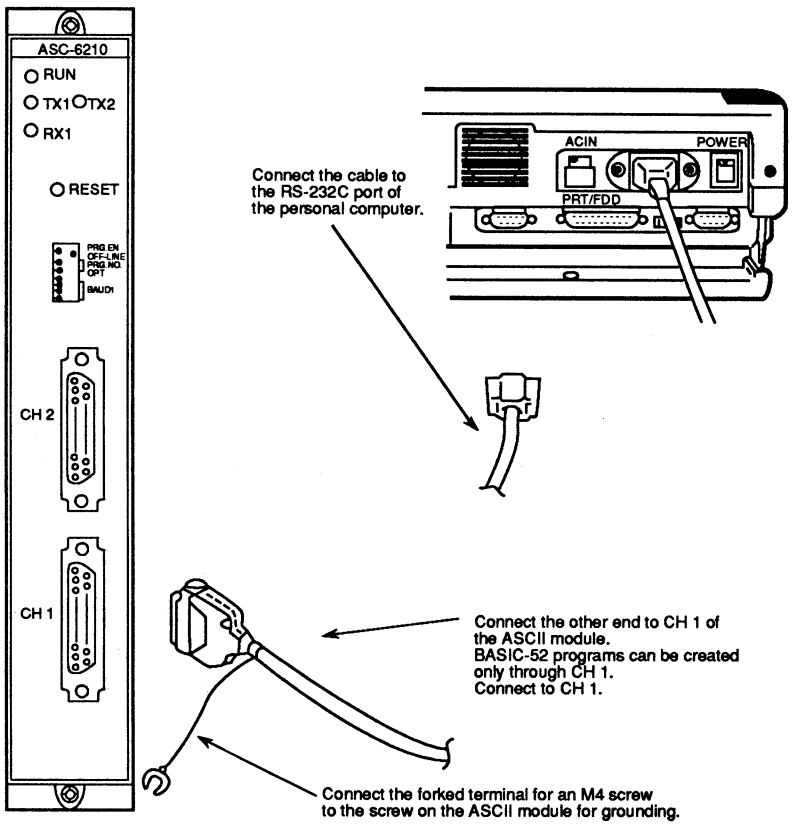
EX CPU
ASCII module
Console device (personal computer or other device)
RS-232C cable
GP, HP, MP, or other programming devices

Step 1: Connect the ASCII module and console device.
Next, create and save a BASIC-52 program.

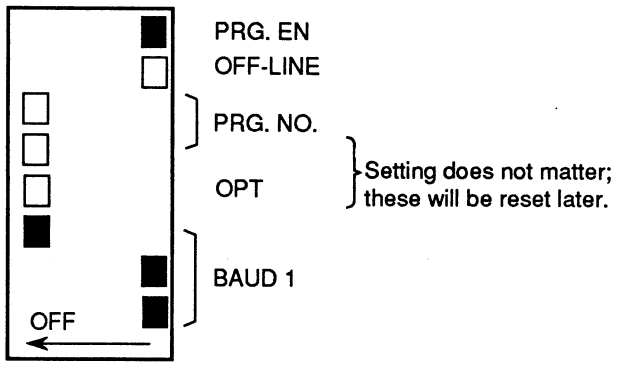
The example described here connects a personal computer that has a BASIC interpreter and the ASCII module. Before connecting the ASCII module, a short BASIC program as shown below is created. Executing this BASIC program changes the personal computer to a console device for the ASCII module. (This program is written in BASIC that functions in the T-3100. Alter as necessary if another model is used.)

```
1000 OPEN "COM1:1200,N,8,1" AS  #1
1010 ON COM(1) GOSUB 1060
1020 COM(1) ON
1030 I$=INKEY$:IF I$<>"" THEN PRINT #1,I$;
1040 GOTO 1030
1050 '
1060 IF LOC(1)=0 THEN RETURN
1070 A$=INPUT$(1,#1)
1080 PRINT A$;:GOTO 1060
SAVE"ASCTERM
Ok
```

Then, connect the ASCII module to the RS-232C connector of a J-3100 series computer as shown below.



Set the DIP switch on the front panel of the ASCII module. BASIC-52 programs are written in the EEPROM enable by the PRG. EN switch, and this switch is turned on. The CH1 baud rate is set so that the RS-232C communication file is opened at a communication baud rate of 1200 bps in the BASIC program "ASCTERM". The DIP switch is set as shown below.

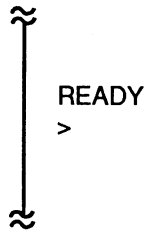


After setting the DIP switch, press the reset switch.

Turn on power to the personal computer to set up the BASIC mode and execute the BASIC program called ASCTERM created earlier. Press the RESET switch on the ASCII module again.

```
LOAD"ASCTERM
Ok

_
LIST
1000 OPEN "COM1:1200,N,8,1" AS  #1
1010 ON COM(1) GOSUB 1060
1020 COM(1) ON
1030 I$=INKEY$:IF I$<>"" THEN PRINT #1,I$;
1040 GOTO 1030
1050 '
1060 IF LOC(1)=0 THEN RETURN
1070 A$=INPUT$(1,#1)
1080 PRINT A$;:GOTO 1060
Ok
RUN

READY
>
```

The display screen of the personal computer should show the message 'READY' and the prompt '>' as shown below.



The display of 'READY' and '>' shows that the ASCII module and personal computer have been connected properly. If they are not displayed, check the cable connection, the BASIC program, and baud rate setting for the ASCII module, then press the RESET switch on the ASCII module again.
Let us start creating BASIC-52 program.

CAUTION Please pay attention to one item. Do not switch change the operation mode select key switch (the RUN/HALT switch of the EX250/500) during editing of a BASIC-52 program. Switching the RUN/HALT switch on the EX CPU from HALT to RUN resets the hardware of the ASCII module and clears the BASIC-52 programs being edited.

Key in the following BASIC-52 program on the keyboard. The program inputting method is nearly the same as that for regular BASIC programming.

```
10      REM SAMPLE WRITE
20      PRINT "SAMPLE WRITE"
30      DIM A(10), B(10)
40      CALL 18
50      POP C
60      PRINT "DATA SENT=", C
70      FOR I=1 TO C
80      CALL 2
90      POP D
100     A(I)=D**2:  B(I)=D**3
110     PRINT USING (######), D, A(I), B(I)
120     NEXT I
130     CALL 14
140     CALL 16
```

HINT The cursor keys do not function normally while the ASCTERM BASIC program is executed. If an input error is made, press the [CR] key and then retype the entire line.

```
READY
>10 REM SAMPLE WRITE
>20 PRINT "SAMPLE WRITE"
>30 DIM A(1),B(10)
>30 DIM A(10),B(10)
>40 CALL 18
```

◄ 'A(1)' is input by mistake instead of 'DIM A(10).'

Check for input errors or other irregularities after inputting all lines. (Input 'LIST [CR]' to list the programs just created on the display screen.)

Input 'SAVE [CR]' if no errors are found in the program.

```
>140 CALL 16

>LIST
10      REM SAMPLE WRITE
20      PRINT "SAMPLE WRITE"
30      DIM A(10),B(10)
40      CALL 18
50      FOR C
60      PRINT "DATA SENT =",C
70      FOR I=1 TO C
80      CALL 2
90      FOR D
100    A(I)=D**2 : B(I)=D**3
110     PRINT USING(######),D,A(I),B(I)
120     NEXT I
130     CALL 14
140     CALL 16


READY
>SAVE  (CR)
```

When the [CR] key is pressed, '1', the 'READY' message, and the '>' mark will be displayed on the CRT screen. The numeral (1) is the program number.

```
140     CALL 16


READY
>SAVE
 1
READY
>
```

The created program has now been written in the EEPROM of the ASCII module. The program position is 'ROM 1.' To check if the program has been correctly saved, input:

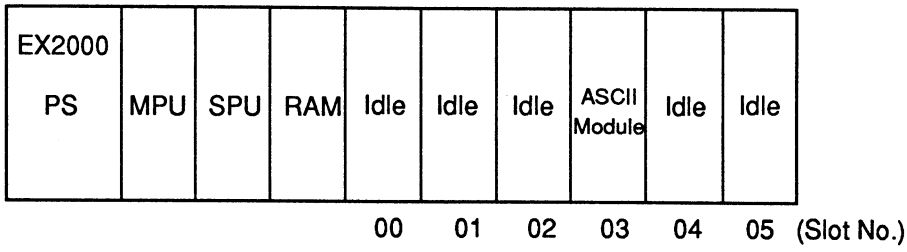
PROM	1 [CR]	(Select the program at ROM 1.)
LIST	[CR]	(List the selected program (ROM 1) on the screen.)

```
140     CALL 16

READY
>SAVE
 1
READY
>PROM1

READY
>LIST
10      REM SAMPLE WRITE
20      PRINT "SAMPLE WRITE"
30      DIM A(10),B(10)
40      CALL 18
50      POP C
60      PRINT "DATA SENT =",C
70      FOR I=1 TO C
80      CALL 2
90      POP D
100    A(I)=D**2 : B(I)=D**3
110     PRINT USING(######),D,A(I),B(I)
120     NEXT I
130     CALL 14
140     CALL 16

READY
>
```

Step 2: Operate the EX CPU to make the I/O allocation.

Next, create the ladder sequence program.

The ladder sequence program for the EX CPU will now be created. Before creating ladder sequence program, we must allocate the ASCII module to the EX CPU. To simplify the explanation, the automatic I/O allocation is selected from the GP control command functions. Assuming that EX2000 is used and that the ASCII module is configured as follows.

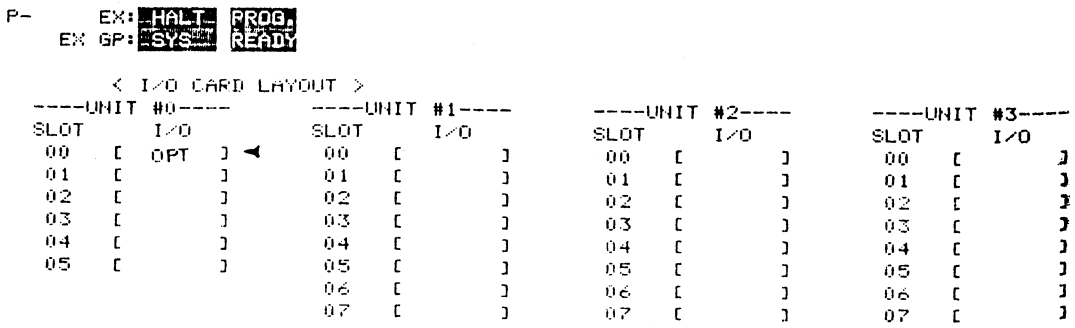


The ASCII module will be assigned as follows after automatic input/output assignment execution when using the ASCII module as an I/O module.

```
P-      EX: HALT  PROG.
     EX GP: SYS   READY

          < I/O CARD LAYOUT >
  ----UNIT #0----      ----UNIT #1----      ----UNIT #2----      ----UNIT #3----
  SLOT     I/O         SLOT     I/O         SLOT     I/O         SLOT     I/O
   00   [ X+Y04W ] ◄    00   [       ]       00   [       ]       00   [       ]
   01   [       ]       01   [       ]       01   [       ]       01   [       ]
   02   [       ]       02   [       ]       02   [       ]       02   [       ]
   03   [       ]       03   [       ]       03   [       ]       03   [       ]
   04   [       ]       04   [       ]       04   [       ]       04   [       ]
   05   [       ]       05   [       ]       05   [       ]       05   [       ]
                        06   [       ]       06   [       ]       06   [       ]
                        07   [       ]       07   [       ]       07   [       ]
```

After executing the command, monitor the I/O allocation information and check that the ASCII module is allocated in the right position of the X+Y04W type.

The ASCII module will be assigned as follows if it is to be used as an option module.



The ASCII module will be assigned as follows if it is to be used as an option module. Then refer to page 30 and after.

The Nos. of the module registers will be as follows if the ASCII module is used as an I/O module.

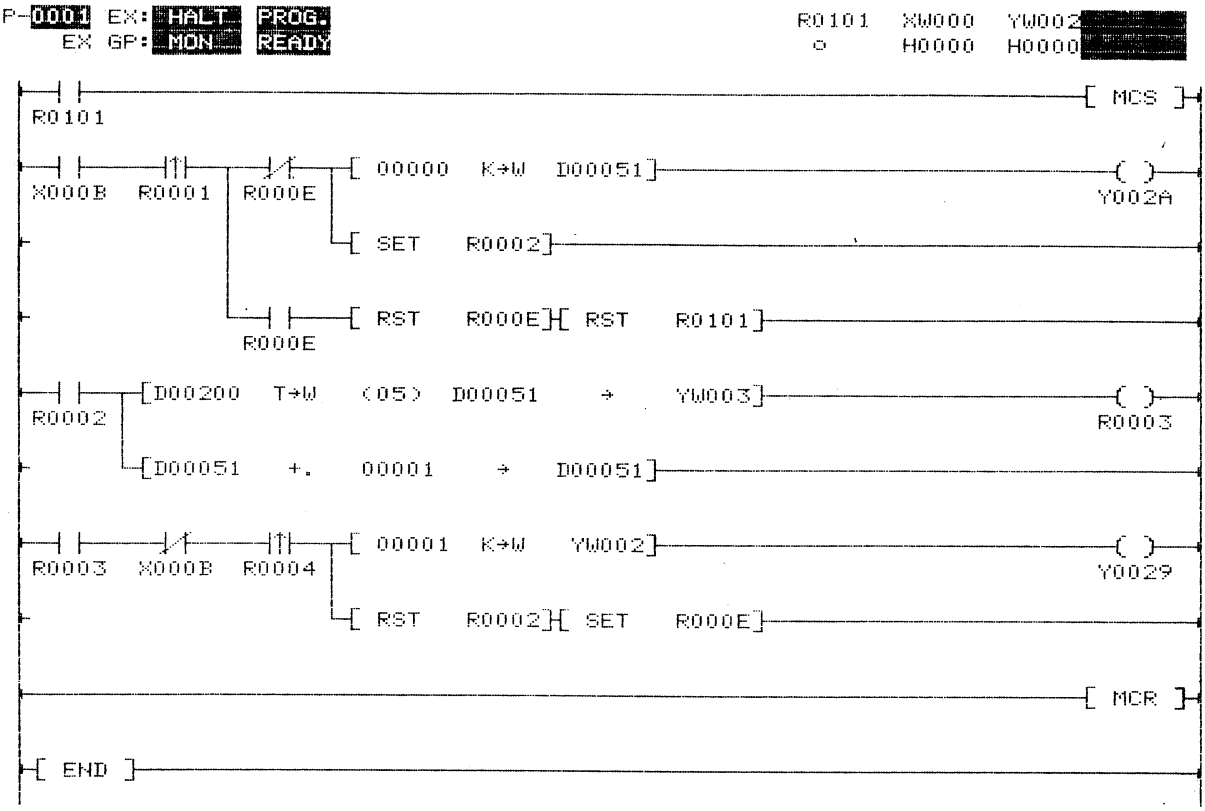
Register No.	Mating Register in ASCII Module
XW000	Status Register
XW001	READ Data Register
YW002	Command Register
YW003	WRITE Data Register

The compositions and functions of the registers such, as the status and READ data registers, will be explained in CHAPTER 5 BASIC READ AND WRITE OPERATIONS.

NOTE

It is possible that I/O assignment has been performed while the internal jumper setting of the ASCII module was still the option module type to the EX CPU, which has no READ/WRITE command if "X 01W" is indicated as the slot No. 00 of Unit #0. Check that the EX CPU version and the internal jumper setting of the ASCII module are correctly matched.

Next, create the following ladder sequence program.



Set the five pieces of data sent by the EX CPU to the ASCII module in the D registers. Enter the Block Monitor mode and set the following values in the five registers D00200 to D00204.

```
P-      EX: HALT  PROG.  D00200
     EX GP: BLKMON READY          KEY IN START NO.

          < STATUS DISPLAY >

    REG.   VALUE  FEDC BA98 7654 3210     REG.   VALUE  FEDC BA98 7654 3210
  D00200   00004  oooo oooo oooo ooo●oo  D00208   00000  oooo oooo oooo oooo
  D00201   00010  oooo oooo oooo ●oo●o  D00209   00000  oooo oooo oooo oooo
► D00202   00020  oooo oooo ooo● o●ooo  D00210   00000  oooo oooo oooo oooo
  D00203   00030  oooo oooo ooo● ●●●o  D00211   00000  oooo oooo oooo oooo
  D00204   00040  oooo oooo oo●o ●ooo  D00212   00000  oooo oooo oooo oooo
  D00205   00000  oooo oooo oooo oooo  D00213   00000  oooo oooo oooo oooo
  D00206   00000  oooo oooo oooo oooo  D00214   00000  oooo oooo oooo oooo
  D00207   00000  oooo oooo oooo oooo  D00215   00000  oooo oooo oooo oooo
```

HINT

Compare these five pieces of data and the data displayed on the console shown on page 21. The data sent by the EX CPU consists of the number of data words (D00200) and the individual data (D00201 to D00204). In this example, the number of data words is four, D00201 to D00204. Thus, '4' is stored in D00200 as the number of data words. The total data pieces sent by the EX CPU will be '5', the sum of the individual data plus one for the size information.

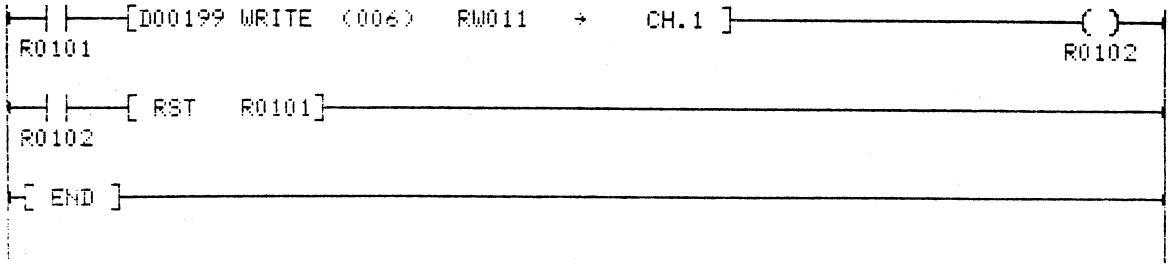
Designate this storage area as the power-failure memory retention area.

The system information will be displayed, the edit mode will be set up, and the D registers will be set to retain memory in a power failure as shown below.

```
P-      EX: HALT  PROG.
     EX GP: SYS   READY

          < SYSTEM INFORMATION >

  1. PROGRAM ID.        [ASCII_SAMP]        9. ERROR STATUS
  2. KEEP AREA TOP   RW[         ] ~ RW999
                     D [  00200] ~ D08191 ◄
                     C [         ] ~ C499
                     T [         ] ~ T499   10. DIAGNOSTIC
  3. MEMORY CAPACITY       16.0K STEPS          SLOT  DIAG. NO.    EVENT
  4. USED PAGE              0001                   1
  5. USED STEP             00048                   2
  6. EX TYPE             EX2000                    3
  7. SYSTEM ID.          EX2000V1.0                4
  8. GP VERSION          GPMP2-V1.1
```

Proceed to step 3 on page 32.

Create the following ladder sequence program to use the ASCII module as an option module.



Sets the data sent by the EX CPU to the ASCII module in the D registers. The block monitor mode will set up and the following data will be set in six registers D00199 to D00204.

```
P-      EX: HALT  PROG.  D00199
     EX GP: BLKMON READY        KEY IN START NO.

          < STATUS DISPLAY >

   REG.  VALUE  FEDC BA98 7654 3210     REG.  VALUE  FEDC BA98 7654 3210
 ▶ D00199 00001  0000 0000 0000 000●   D00207 00000  0000 0000 0000 0000
   D00200 00004  0000 0000 0000 0●00   D00208 00000  0000 0000 0000 0000
   D00201 00010  0000 0000 0000 ●0●0   D00209 00000  0000 0000 0000 0000
   D00202 00020  0000 0000 000● 0●00   D00210 00000  0000 0000 0000 0000
   D00203 00030  0000 0000 000● ●●●0   D00211 00000  0000 0000 0000 0000
   D00204 00040  0000 0000 00●0 ●000   D00212 00000  0000 0000 0000 0000
   D00205 00000  0000 0000 0000 0000   D00213 00000  0000 0000 0000 0000
   D00206 00000  0000 0000 0000 0000   D00214 00000  0000 0000 0000 0000
```

HINT

Compare these six groups of data and the data displayed on the console shown on page 21. The data of the first register (D00199) of the data table will be the command that the EX CPU sends to the ASCII module, rather than the data sent by the EX CPU. The BASIC-52 program was saved in "ROM1" when the BASIC-52 program was saved in the ASCII module in Step 1. Therefore, set "1" in D00199 to request start of the BASIC-52 program of ROM1 by the EX CPU. Registers D00200 and after contain data sent by the EX CPU. The data consists of the number of data words (D00200) and the data (D00201 to D00204) sent by the EX CPU. In this example, the number of data words sent by the EX CPU is four, from D00201 to D00204. Thus, "4" is stored in D00200 as the number of data words.

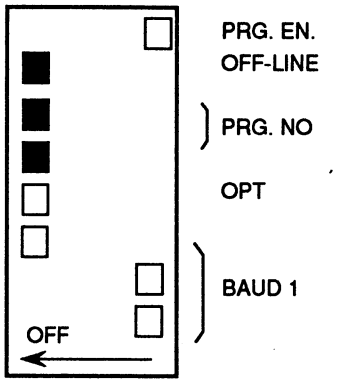
Designate this storage area as the power-failure memory retention area.

The system information will be displayed, the edit mode will be set up, and shown below. The D registers will be set retain memory in a power failure as shown below.

```
P-      EX: HALT  PROG.
     EX GP: SYS   READY

          < SYSTEM INFORMATION >

  1. PROGRAM ID.        [ASCII_SAMP]          9. ERROR STATUS
  2. KEEP AREA TOP  RW[      ] ~ RW999
                    D [  00199] ~ D08191 ◀
                    C [      ] ~ C499
                    T [      ] ~ T499   10. DIAGNOSTIC
  3. MEMORY CAPACITY        16.0K STEPS        SLOT  DIAG. NO.   EVENT
  4. USED PAGE               0001                1
  5. USED STEP               00013               2
  6. EX TYPE                EX2000               3
  7. SYSTEM ID.             EX2000V2.1           4
  8. GP VERSION             GPAP2-V1.1
```

Step 3: Reset the ASCII module DIP switch.

The BASIC-52 program and ladder sequence program were created in steps 1 and 2. The DIP switch on the front panel of the ASCII module must now be reset to allow the ASCII module to interface with the EX CPU. Set the DIP switch as shown below.



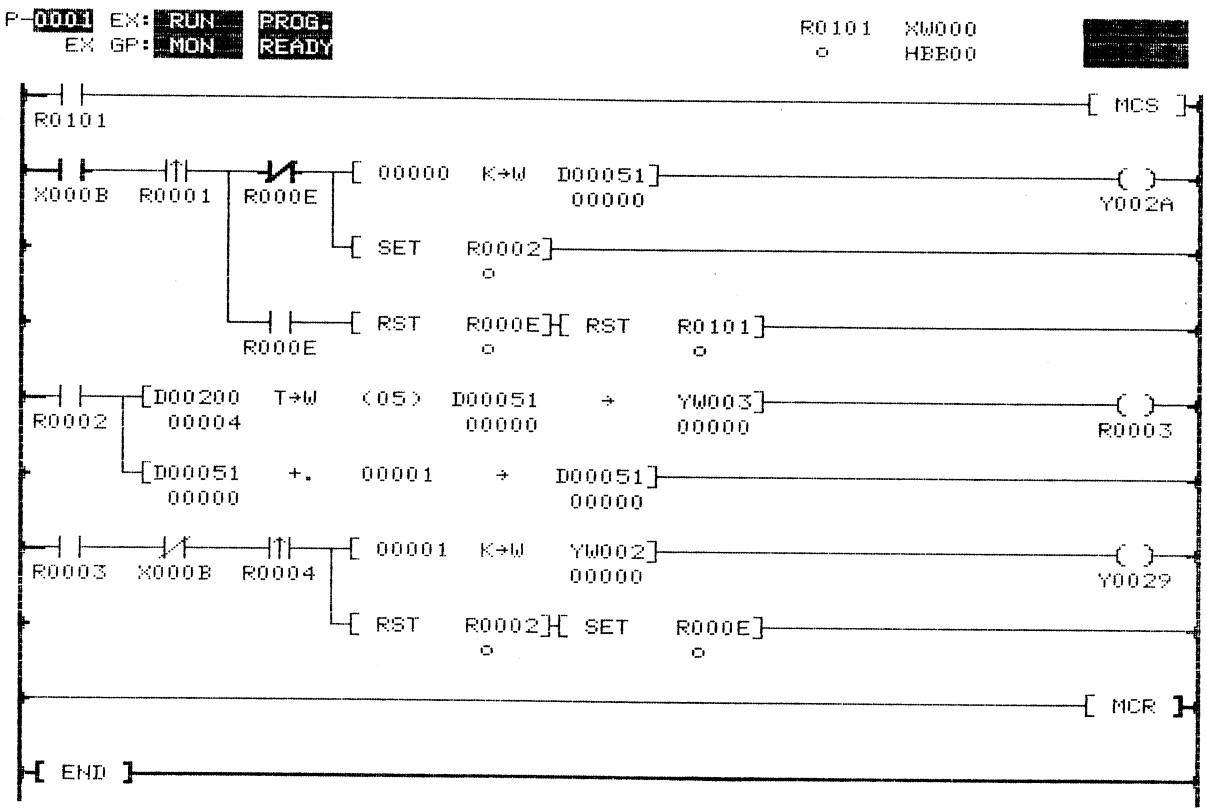
After setting the DIP switch, press the RESET switch.

Then proceed to the next page if the ASCII module is used as an I/O module. Proceed to page 35 if the ASCII module is used as an option module.

Step 4: Execute the ladder sequence program.

Set the operation mode select key switch on the EX CPU (RUN/HALT switch of the EX250/500) to the RUN position.

This ladder sequence program cannot be started unless R0101 of the device is set to ON. Before setting R0101 to ON, check the value of XW000 (ASCII module status register). Register R101 and XW000 in the auxiliary data display area in the GP monitor mode. Is the value of XW000 "HBB00"?

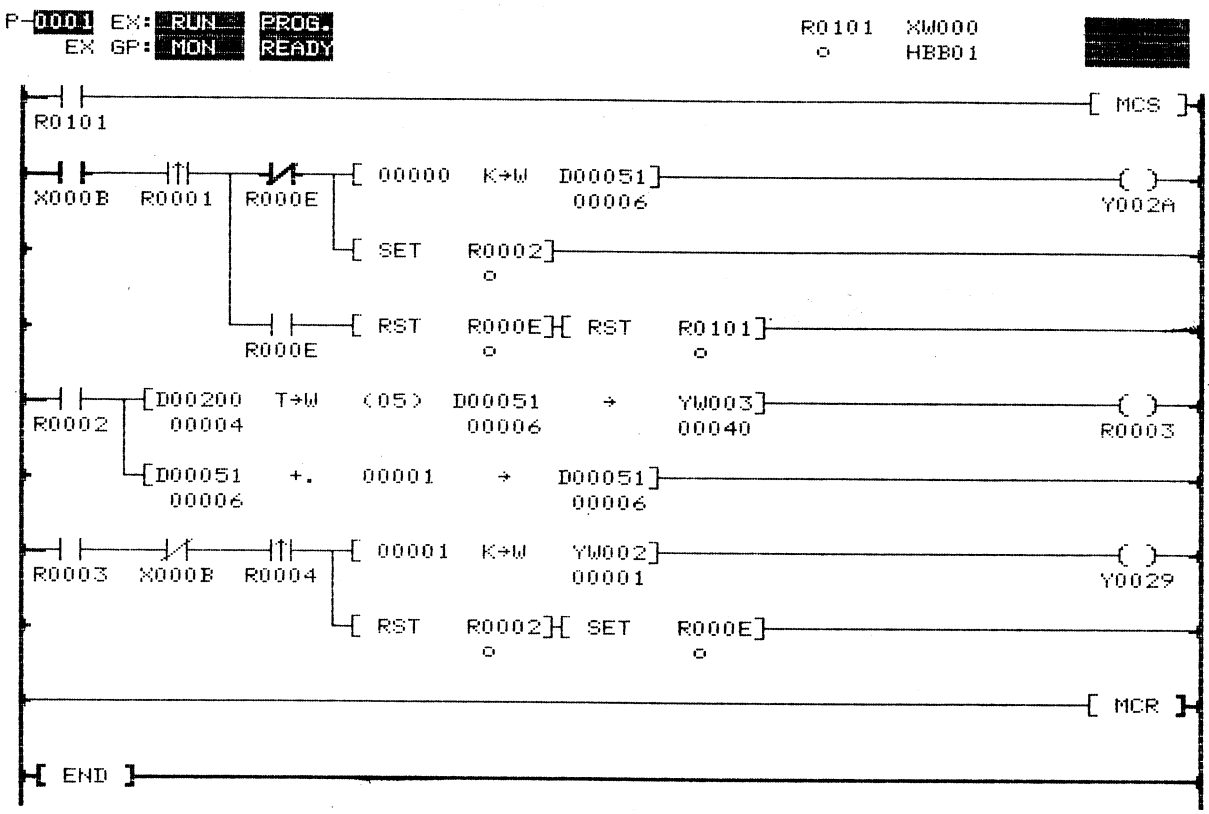


If 'HBB00' is not shown, check the DIP switch setting and press the RESET switch again.

Set R0101 to ON. The following message will then be shown on the screen of the personal computer connected to the ASCII module.

```
SAMPLE WRITE
DATA SENT = 4
     10      100     1000
     20      400     8000
     30      900    27000
     40     1600    64000
```

The entire ladder sequence program should be executed, and device R0101 should be reset to OFF on the EX CPU side.



In order to execute this sample program again, set device R0101 to ON again.

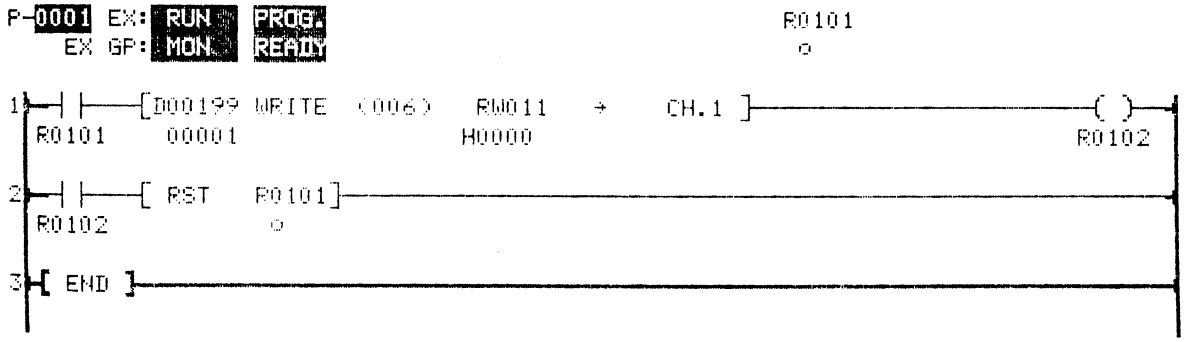
Set the operation mode select key switch on the EX CPU (RUN/HALT switch of EX250/500) to the RUN position.

This ladder sequence program cannot be started unless R0101 of the device is set to ON. Register R0101 in the auxiliary data display area by the GP monitor mode, and then set R0101 to ON.

The following message will be shown on the CRT screen of the personal computer connected to the ASCII module:

```
SAMPLE WRITE
DATA SENT = 4
     10     100    1000
     20     400    8000
     30     900   27000
     40    1600   64000
```

The entire ladder sequence program should have been executed, and R0101 of the device should be reset to OFF on the EX CPU side.



Set device R0101 to ON to execute this sample program again.

5. Basic Read and Write Operations

This chapter describes the following two basic operation modes, which are the basis for interfacing between the ASCII module and EX CPU:

- Basic READ operation
- Basic WRITE operaiton

This chapter describes operation of the ASCII module for use as an I/O module and as an option module. Refer to:

section 5.1 for use of the ASCII module as an I/O module
section 5.2 for use of the ASCII module as an option module.

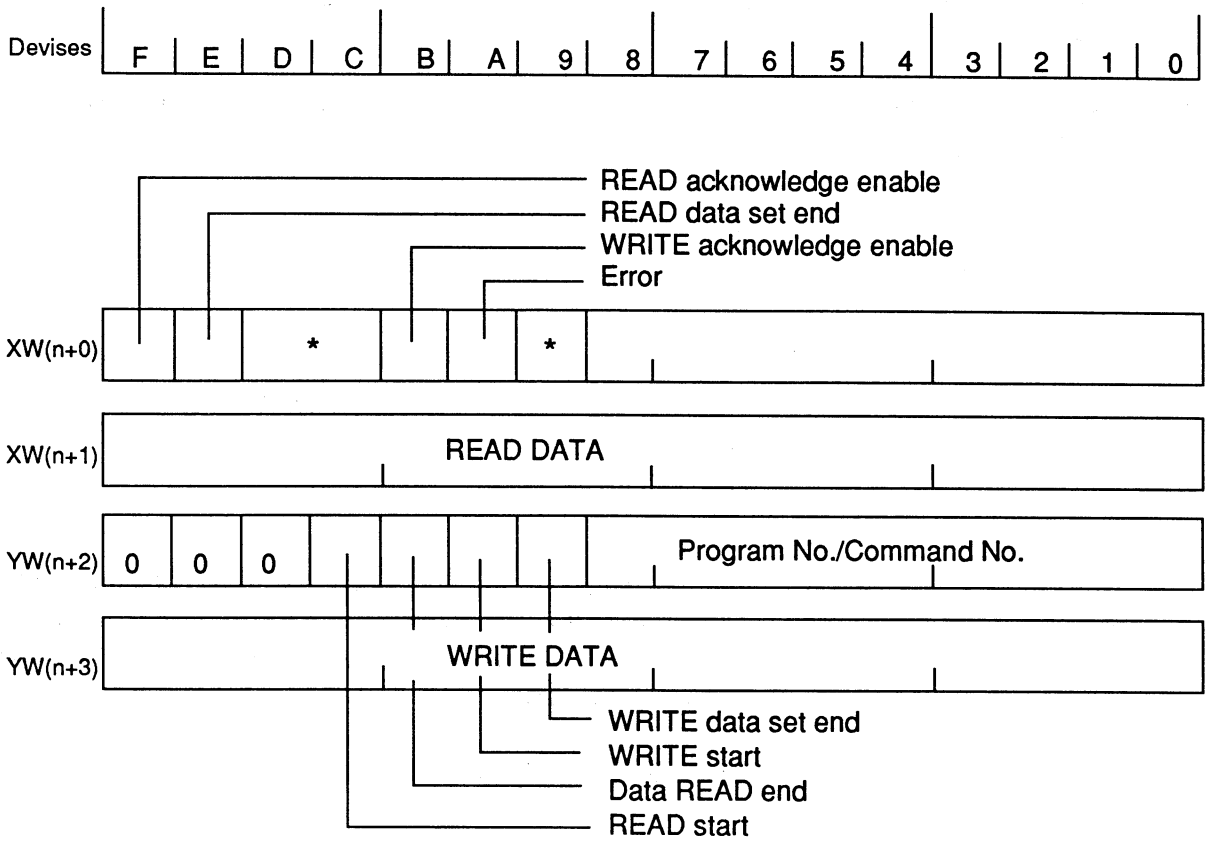
5.1 Basic Operation as I/O Module

This section describes basic two processing operations when the ASCII module is used as an I/O module. First, the register composition of the ASCII module is shown. Then, both READ and WRITE operations are described using programming examples of the EX CPU.

5.1.1 Register assignment

The ASCII module type is X+Y04W if a ladder sequence program is created without using the READ/WRITE commands (using the ASCII module as an I/O module). The register configuration and the function of the each device making up the registers are described in the following pages.

Figure 5.1 shows the register configuration.



* : Not used.

Figure 5.1 Register Configuration of the ASCII Module

- Status Register XW(n+0)

Register XW(n+0) shows the status of the ASCII module.

The devices in this register act as flags to show the status of the ASCIi module when the ASCII module and EX CPU are interfaced.

Device	Name	Function
F	READ acknowl-edge enable	This device shows whether or not the ASCII module can accept the READ sequence. If the ASCII module can accept the READ sequence (that is, if the READ channel is in IDLE status), the ASCII module sets this device to '1.'
E	READ data set end	This device shows whether or not the EX CPU could read data from the ASCII module. The ASCII module executes the corresponding BASIC-52 program after acknowledging the READ sequence. If data can be read by the EX CPU after executing the program (if the answer from the ASCII module is 'DATA IS READY'), the ASCII module sets this device to '1.'
B	WRITE acknowl-edge enable	This device shows whether or not the ASCII module can acknowledge the WRITE sequence. If the ASCII module can accept the WRITE sequence (that is, if the WRITE channel is in IDLE status), the ASCII module sets this device to '1.'
A	ERROR	This device shows whether or not the ASCII module has detected an error. If the ASCII module finds an error when starting or executing a program, it sets this device to '1.' The ASCII module sets the error code in accordance with the error type in the XW(n+1) register as the READ data to the EX CPU. The ASCII module reacts differently depending on the type of error that has occurred. Refer to APPENDIX 4.3 Interfacing Errors between ASCII Module and EX CPU for details of the corresponding recovery operations.

Error Code List

Error codes are set in the lower byte of register XW(n+1).

Error Code	Error	Error Description	Operation of the ASCII module after setting error codes
06H	CPU Error	Error detected in CPU of ASCII module.	After setting this error code, the ASCII module resets it self. If the module can not be reset, the RUN LED lamp blinks.
07H	RAM Error	RAM error detected in the ASCII module.	
08H	ROM Error	ROM error detected in the ASCII module.	
09H	WD Timer Error	Watchdog timer error detected.	After setting this error code, the ASCII module resets it self. Refer to APPENDIX 5. description of the special function IE, for more information.
0AH	Program No. Error	Illegal Program No. sent by EX CPU (not 0<n≤existing program No.).	After setting this error code the ASCII module again sets '1' to the READ and WRITE acknowledge enable devices so that commands from the EX CPU can be received.
0BH	Illegal Command Error	EX CPU has set READ START and READ END simultaneously. Illegal command No. sent by EX CPU.	

Error Code	Error	Error Description	Operation of the ASCII module after setting error codes
0CH	Reset Routine Execution Error	ASCII module is executing reset routines, such as self-diagnosis and module initialization, and can not acknowledge commands from EX CPU.	After setting this error code, the ASCII module resets itself.

- Read Data Register XW(n+1)

  Data read by the EX CPU is set in Register XW(n+1). The ASCII module stores data to be sent to the EX CPU in a place called the READ buffer memory inside it. The EX CPU reads this data through this register. The ASCII module can set a maximum of 500 words in the READ buffer memory at one time.

  An error code matching the error nature will be set if an error occurs.

- Command Register YW(n+2)

Register YW(n+2) sets commands received from the EX CPU. A command received from the EX CPU during interfacing between the ASCII module and EX CPU is set in each device of this register.

The names and functions of the devices making up the command register are listed below.

Device	Name	Function
C	READ Start	The EX CPU sets this device to '1' and simultaneously sends the BASIC-52 program number for READ sequence. Thus, the EX CPU requests the ASCII module to start program execution.
B	Data READ End	The EX CPU sets this device to '1' when it finishes reading READ data. Thus, the EX CPU notifies the ASCII module that it has finished reading data.
A	WRITE Start	The EX CPU sets this device to '1' when it starts writing data to the ASCII module during WRITE sequence. Thus, the EX CPU notifies the ASCII module that data writing has been started. The ASCII module can be reset by setting this device and WRITE data set end device simultaneously during a ladder sequence program.
9	WRITE Data Set End	The EX CPU sets this device to '1' when it finishes writing WRITE data. The BASIC-52 program No. for WRITE sequence is also designated at this time. Thus, the EX CPU requests the ASCII module to start execution of the specified program.
8 | 0	Program No. or Command	A No. is set in this area when the EX CPU designates the specific program to be started, or the specific command to be sent. Program Nos. 1 to 255 and command Nos. 256 to 511 are effective.

The commands sent by the EX CPU to the ASCII module are listed below.

Command No.	Command
261	The EX CPU requests the ASCII module to reset.

- Write Data Register YW(n+3)

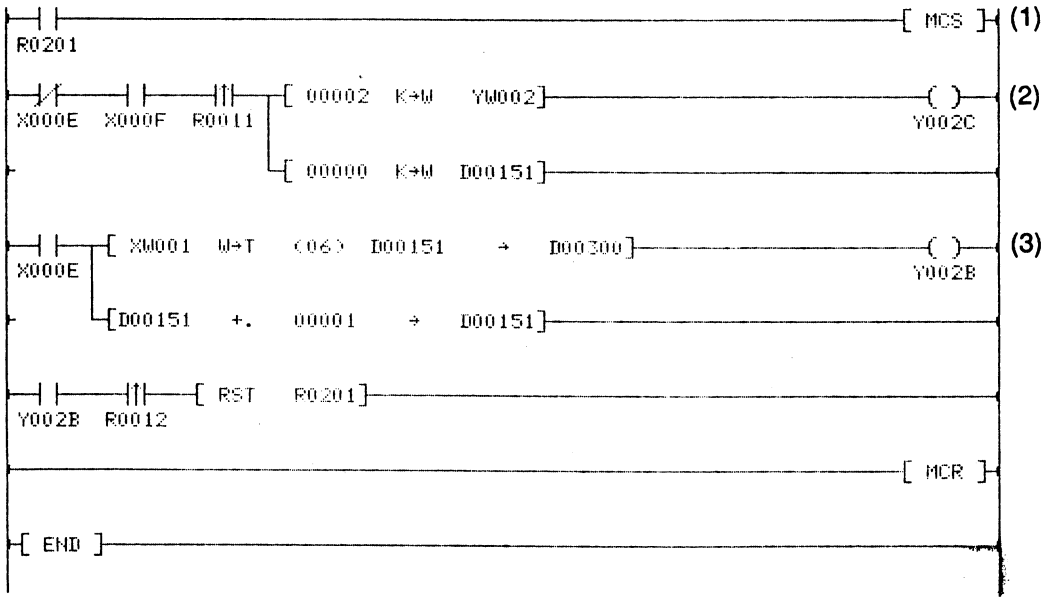
Register YW(n+3) sets data to be written in the ASCII module by the EX CPU. The ASCII module stores data receivedd from the EX CPU through this register internally in its WRITE buffer memory.

The EX CPU can send a maximum of 500 words to the ASCII module at a time.

5.1.2 Basic READ operation

This section outlines the basic READ operation using a ladder sequence program example.

The following ladder sequence program will be used as an example.



The functions of the devices and registers are shown below.

Register No.	Function
XW000	Status register
XW001	READ data register
YW002	Command register
D00151	Data storage area pointer
D00300 | D00305	Area to store data read from ASCII module.

Device	Function
X000E	READ data set end
X000F	READ acknowledge enable
Y002B	Data READ end
Y002C	READ start
R0011	Differential contact to switch on READ start device by 1 scan time.
R0012	Differential contact to switch on data RAED end device by 1 scan time.
R0201	READ sequence being executed.

This sequence program starts the program of ROM2 on the ASCII module side.

The timing chart of this sample program is shown on page 45. Read the outline of processing on the next page comparing the sample program, timing chart, and functions of the ASCII module registers and devices.

(1) Start the READ sequence program by setting device R0201 READ Sequence Being Executed to ON.

(2) Send the ROM program No. to be started by the ASCII module side and set the READ start device simultaneously if the READ Data Set End Device and READ Acknowledge Enable Device of the ASCII module are OFF and ON, respectively.

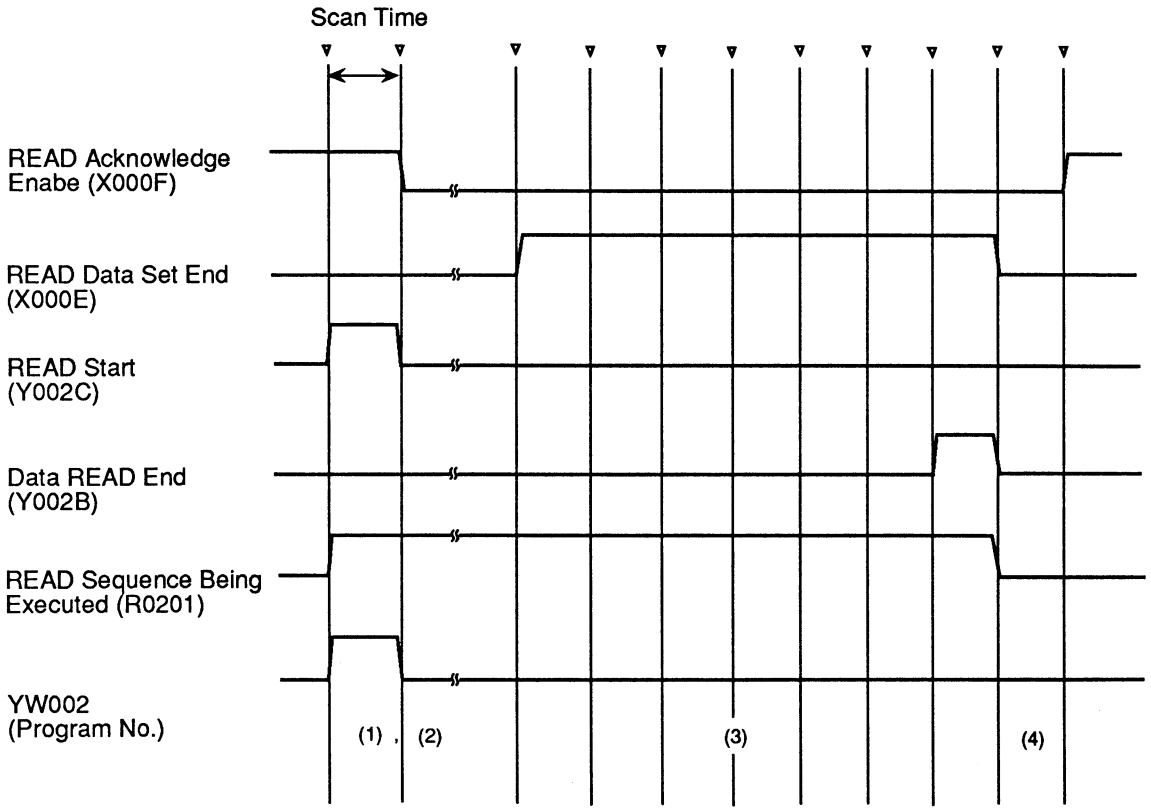
The ASCII module executes a BASIC-52 program and sets data read by the EX CPU in the READ buffer memory when the READ start device is set.

The ASCII module sets the READ Data Set End Device to "1" when it finishes setting all data to be read.

(3) The EX CPU reads data when the READ Data Set End Device of the ASCII module switches on. The EX CPU sets the Data READ End Device when all data is read.

Reading for one cycle ends when data rading is complete. The READ Sequence Being Executed Device is also reset at this time.

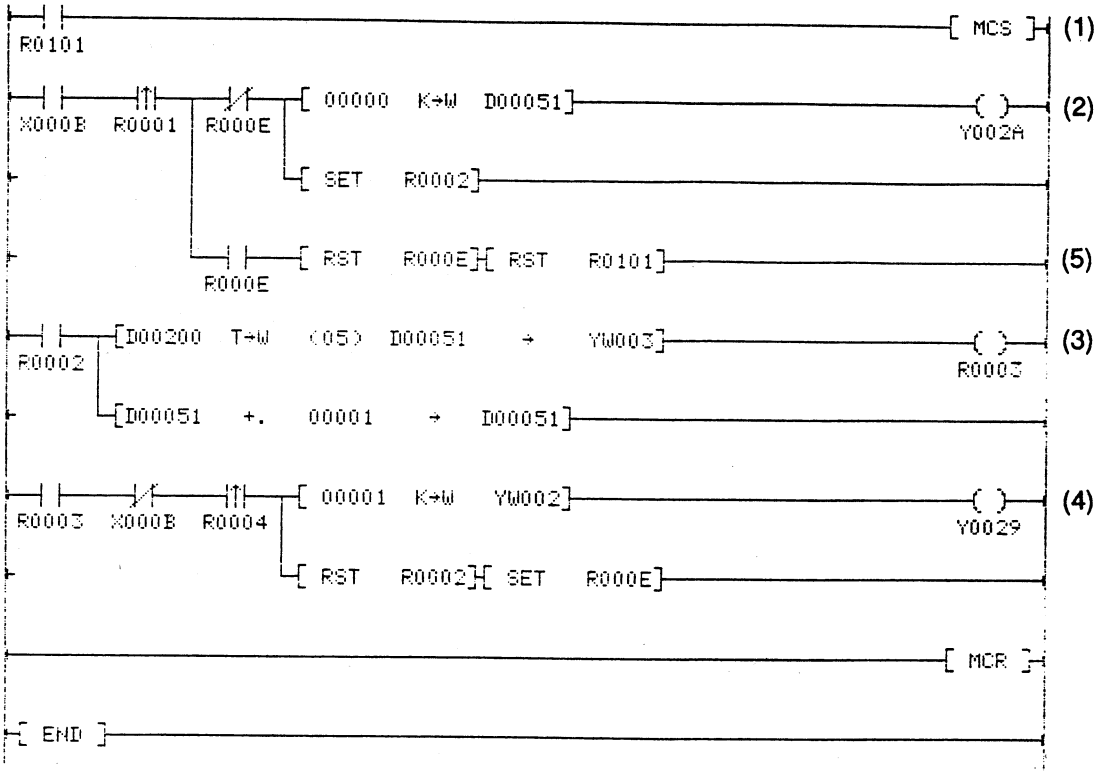
(4) The ASCII module knows that the EX CPU has read all the data when the Data READ End Device is set. The ASCII module resets READ Data Set End Device and resets the READ Acknowledge Enable Device to "1."



5.1.3 Basic WRITE operation

This section outlines the basic WRITE operation using a ladder sequence program example.

The sample ladder sequence program used in chapter 4 (shown below) will be used.



The functions of the devices and registers are given below.

Register No.	Function
XW000	Status register
YW002	Command register
YW003	WRITE data register
D00051	Data storage area pointer
D00200 | D00204	Storage area for data sent to the ASCII module.

Device	Function
X000B	WRITE acknowledge enable
Y0029	WRITE data set end
Y002A	WRITE start
R0001	Differential contact to switch on WRITE start device by 1 scan time
R0002	Data WRITE being executed
R0003	Data WRITE end
R0004	Differential contact to switch on WRITE data set end device by 1 scan time
R000E	BASIC-52 program being executed
R0101	WRITE sequence being executed

This sequence program starts the program of ROM1 on the ASCII module side.

The timing chart of this sample program is shown on page 48. Read the outline of processing in the next page comparing the sample program, timing chart, and functions of the ASCII module registers and devices.

(1) Start the WRITE sequence program by setting device R0101 WRITE Sequence Being Executed to on.

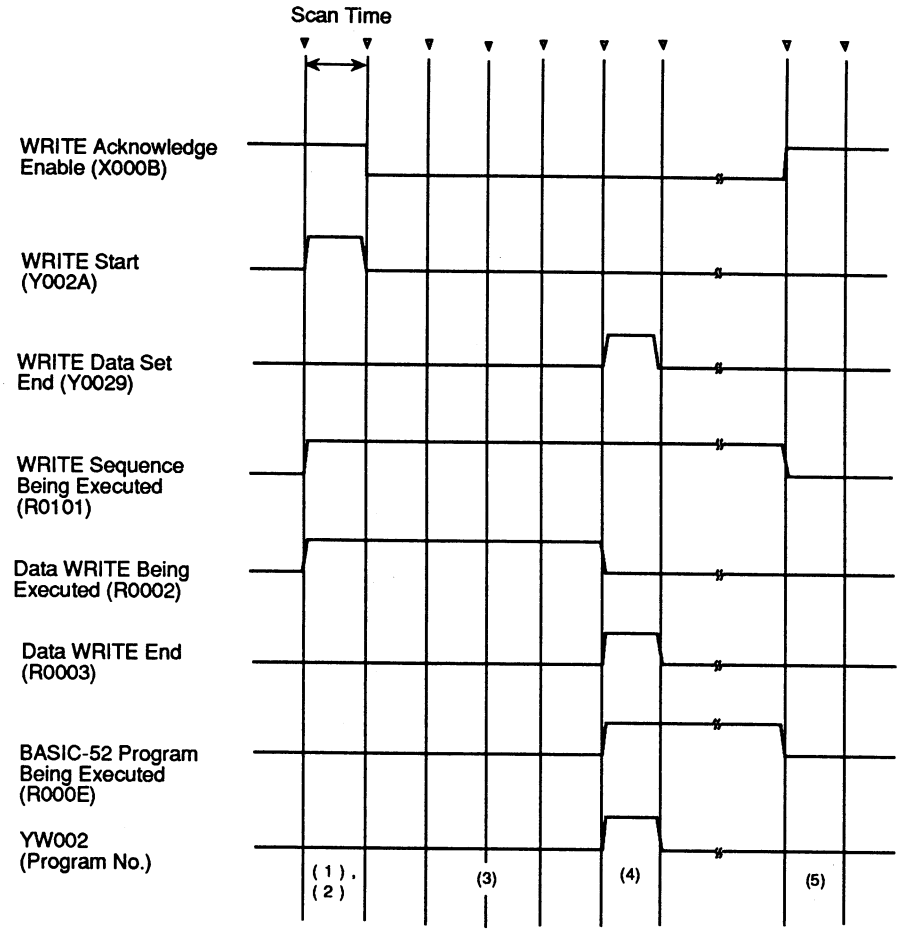
(2) Set the WRITE Start Device if the WRITE Acknowledge Enable Device of the ASCII module is ON. Also set the Data WRITE Being Executed Device.

(3) The EX CPU continues to write data with the ASCII module while the Data WRITE Being Executed Device is set. The data will be stored in the WRITE buffer memory inside the ASCII module. After all data is written, the Data WRITE End Device is set.

(4) After data writing is finished, the No. of the ROM program to be activated on the ASCII module side is sent. The WRITE Data Set End Device is also set simultaneously. Data written in the WRITE Data Register will not be stored in the WRITE buffer memory inside the ASCII module after the WRITE Data Set End Device has been set. The Data WRITE Being Executed Device is reset.

The ASCII module starts executing the BASIC-52 programs when the WRITE Data Set End Device is set. The BASIC-52 Program Being Executed Device will then be set.

(5) After finishing processing the BASIC-52 program, the ASCII module sets the WRITE Acknowledge Enable Device to "1" to show that commands from the EX CPU can be accepted again. The BASIC-52 program Being Executed Device is reset when this device switches on again. This ends WRITE processing for one cycle and the WRITE Sequence Being Executed Device is also reset.



5.2 As Option Module

This section describes two basic processing operations when the ASCII module is used as an option module: the input/output assignment method when the ASCII module is used as an option module, and the READ/WRITE command, which is a command special to the ASCII module.

5.2.1 I/O allocation as option module

Channel Nos. replace the input/output areas, such as XW and YW, when the ASCII module is used as an option module. If several ASCII modules are used, the modules are differntiated by channel Nos.

The method of controlling the ASCII modules by the EX CPU using channel Nos. is described. As an example, assume that I/O modules including ASCII modules are mounted in the EX CPU as follows.

EX CPU (EX250/500 CPU or EX 2000 CPU)	16 Points/ 1 Register Input Module	ASCII Module CH2 CH1	64 Points/ 4 Register Input Module	ASCII Module CH2 CH1	32 Points/ 2 Register Output Module

The following input and output allocation is made:

Slot No.	I/O Type	Register No.
0	X 01 W	XW 00
1	OPT	———
2	X 04 W	XW01-XW02-XW03-XW04
3	OPT	———
4	Y 02W	YW05-YW06

Channel numbers are allocated to the ASCII modules mounted in slots 1 and 3 as follows:

EX CPU	Input Module	ASCII Module CH2 CH1	Input Module	ASCII Module CH4 CH3	Output Module

Be careful not to mix CH1 and CH2 displayed on the front panel of the ASCII module and the channel numbers managed by the EX CPU.
An EX CPU can mount a maximum of four ASCII modules as option modules.

EX CPU	ASCII Module CH2 CH1	ASCII Module CH4 CH3	ASCII Module CH6 CH5	ASCII Module CH8 CH7

The diagram above shows the allocation of channel numbers when the maximum number of ASCII modules are mounted.

5.2.2 READ/WRITE commands

The EX CPU versions that can use the READ/WRITE commands allow mounting of ASCII modules as option modules. This enables easier interface between the EX CPU and ASCII module than the conventional ladder sequence method.

The two dedicated commands for interfacing between the EX CPU and ASCII module, READ and WRITE, are described in the following pages.

READ	ASCII Read (FUN 098)

Expression

Condition Input — CH.Ⓐ READ [nnn] Ⓑ → Ⓒ — Completion/ Error Output

H

Data Displayed

Number of Steps
5

Function

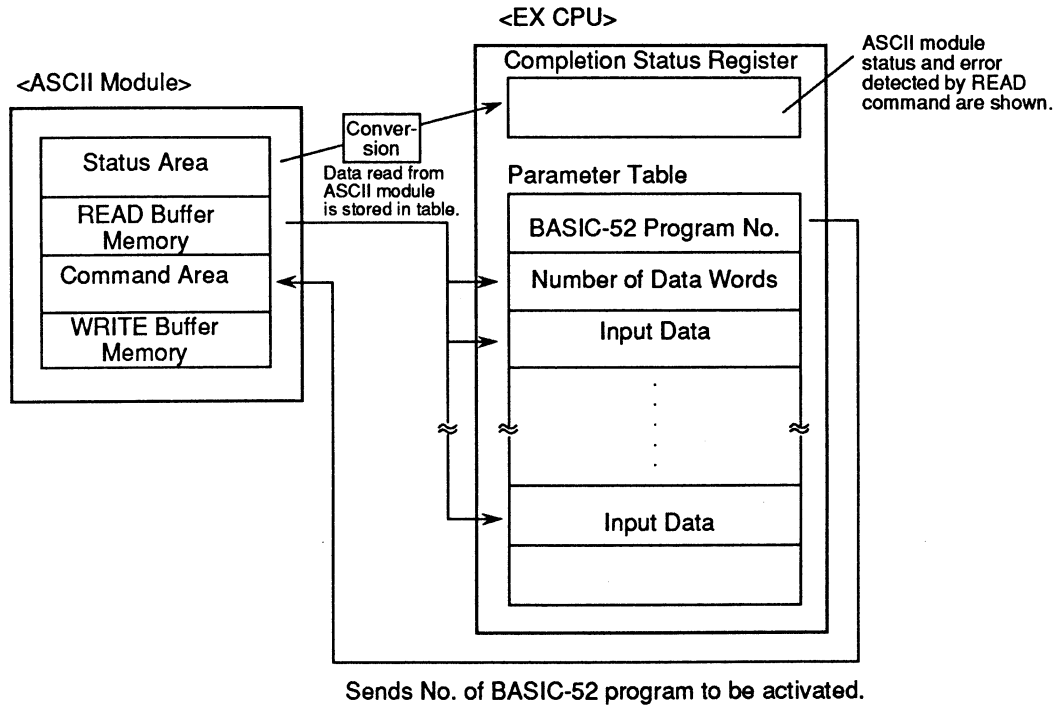
Make ASCII module of CH Ⓐ run the BASIC-52 program number specified by Ⓒ and read data of size shown by Ⓒ+1 in registers begining with the register shown by Ⓒ +2.
Program completion status is shown by Ⓑ.

Input condition	Program		Output
OFF	No execution		OFF
ON	Execution	Executing	OFF
		Completion or (Error)	ON

Operand

Symbol	Name	R	X	Y	Z	RW	XW	YW	ZW	D	T	C			Quantity
A	Channel No.														1~8
nnn	Table Size														2~502(EX2000)
															2~252(EX250/500)
B	Completion Status Register					○		○	○						
C	First Register of Parameter					○		○	○	○					

Operation Diagram



Explanation:

Read the following while referring to the operation diagram shown on the preceding page and the timing chart shown below.

The EX CPU executes the READ command when the condition input is ON and processes as follows.

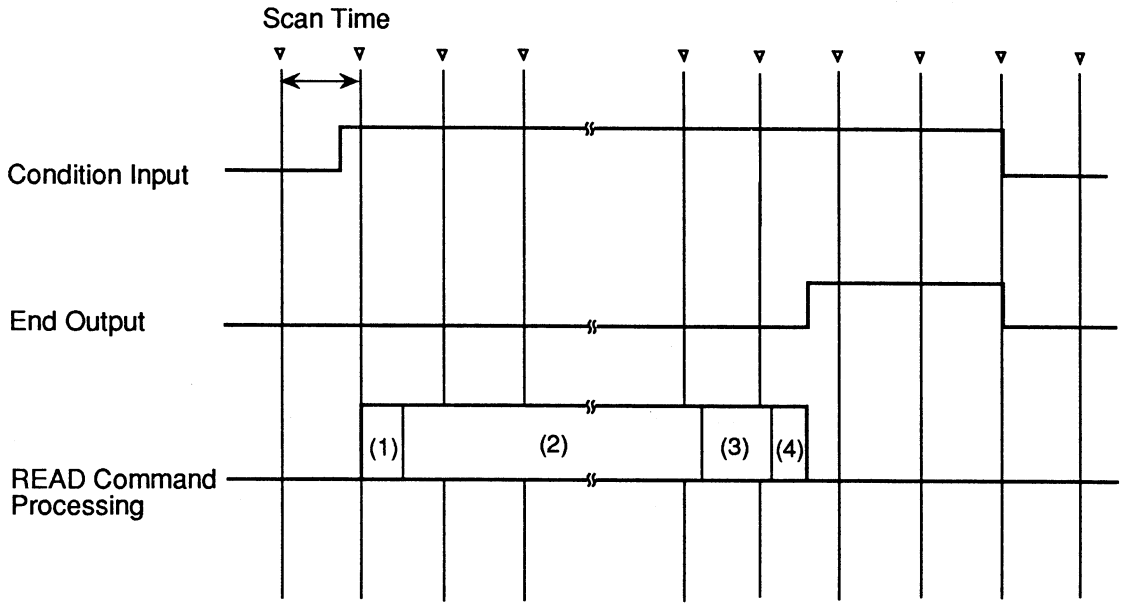
(1) The EX CPU transfers the BASIC-52 program No. to the ASCII module designated by the channel No., and sends a request to activate the program of this No.

(2) The ASCII module executes the BASIC-52 progam and sets the data to be read by the EX CPU in its internal READ buffer memory.

(3) The EX CPU reads the data set in the READ buffer memory and stores it in the input data area of the parameter table.

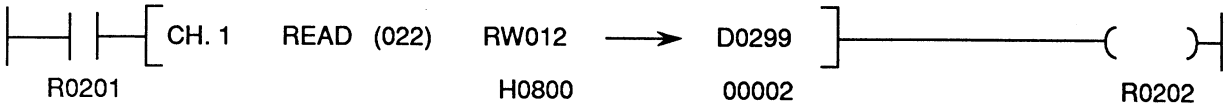
(4) The READ command is finished when all data is read.

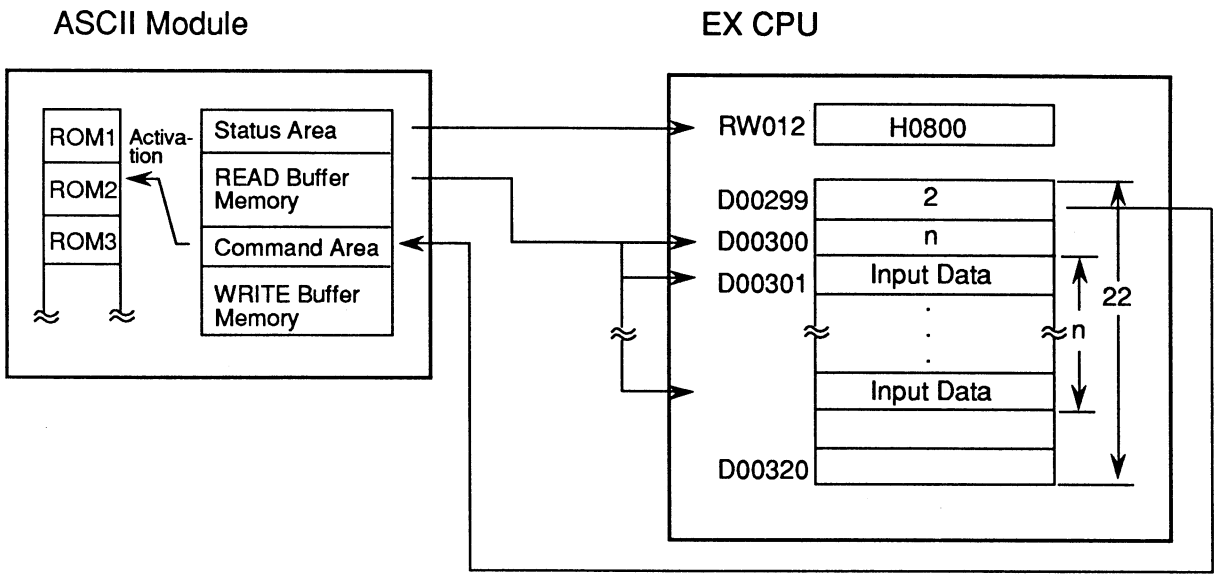
Refer to page 57 and after for the details of the Completion Status Register data.



READ Command Timing Chart

Program Example





- Executes the BASIC-52 program of the No. designated by the data of Register D00299 against the ASCII module shown by CH.1 when Normally-Open Contact R0201 is ON.

- Reads the data set on the ASCII module after the ASCII module executes the designated program. Stores the input data size in the table beginning with D00300 and the input data in the table beginning with D00301.

- The completion status will be shown in RW012.

Note: • Maintain the condition input while the ASCII module is processing.

- Error-end results without activating the ASCII module if the channel No. and table size are illegal.

- Reads data only within the table range if the number of words surpasses the table size and finishes operation.

WRITE	ASCII Write (FUN 099)

Expression

Condition Input — (A) WRITE [nnn] (B) → CH.(C) — Completion/ Error Output

H

Data Displayed

Number of Steps
5

Function

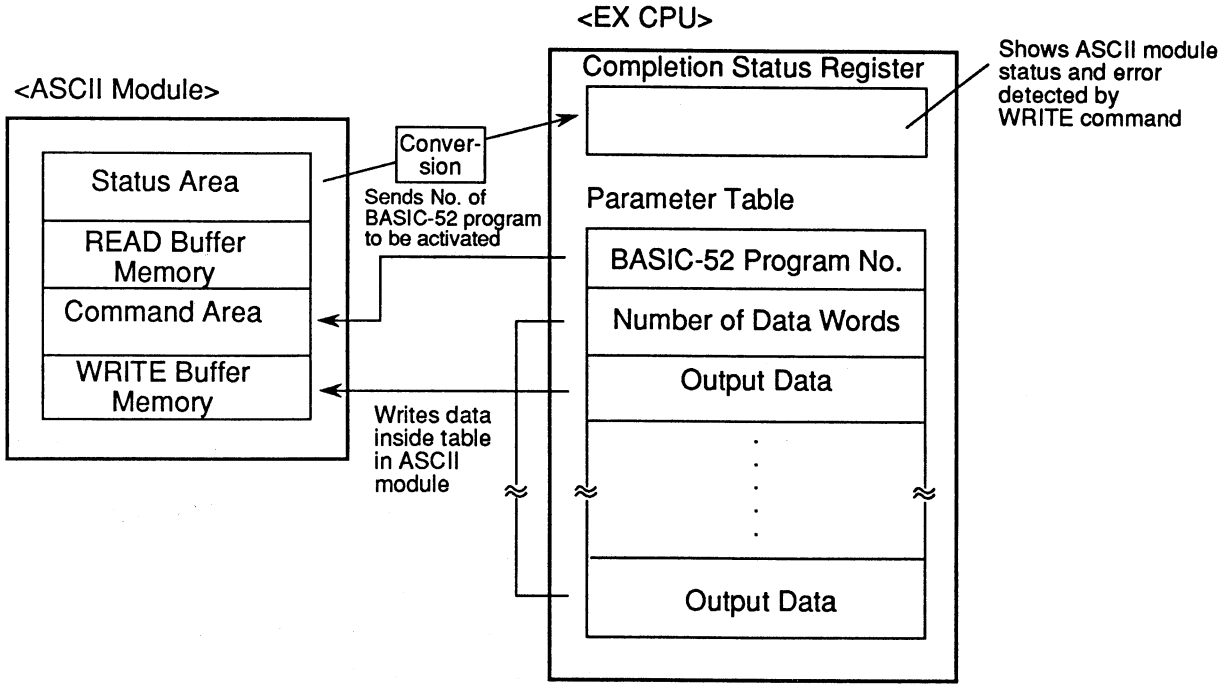
After outputting the data of (A) + 1 size starting from register (A) + 2 to ASCII module indicated by channel (C), the BASIC-52 program indicated by number (A) is executed.
The program completion status is indicated by (B).

Input condition	Program	Output
OFF	No execution	OFF
ON	Execution: Executing	OFF
	Execution: Completion or (Error)	ON

Operand

Symbol	Name	R	X	Y	Z	RW	XW	YW	ZW	D	T	C			Quantity
A	First Register of Parameter					○		○	○	○					
nnn	Table size														2~502(EX2000)
															2~252(EX250/500)
B	Completion Status Register					○		○	○						
C	Channel No.														1~8

Operation Diagram



Explanation:

Read the following while referring to the operation diagram shown on the preceding page and the timing chart shown below.

The EX CPU executes the WRITE command when the condition input is ON, and processes as follows.

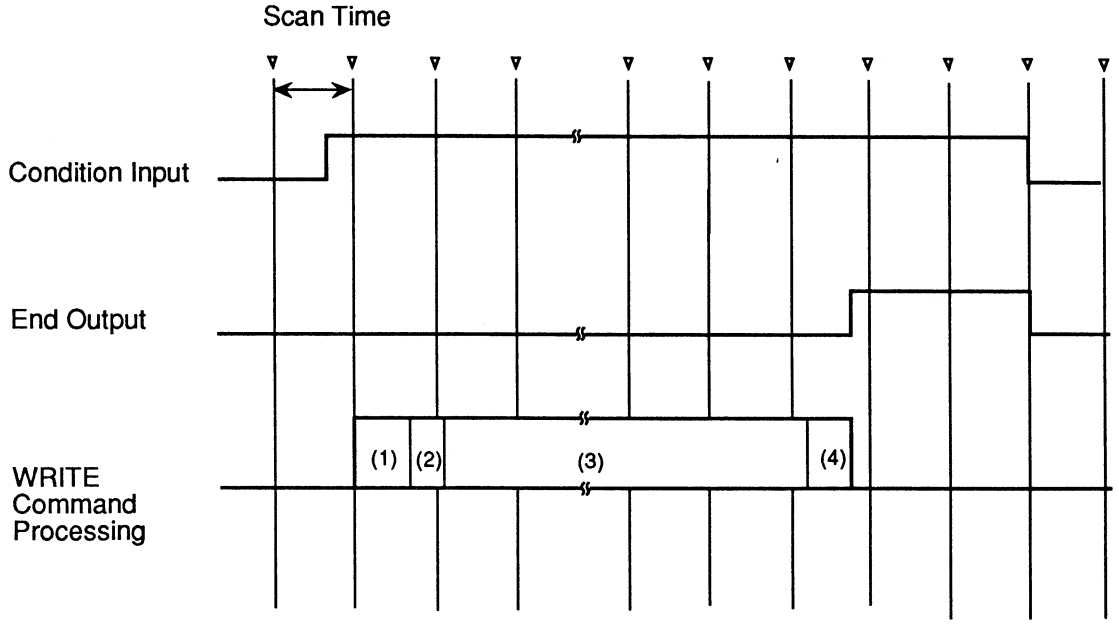
(1) The EX CPU transfers the data stored in the output data area of the parameter table to the ASCII module designated by the channel No. The ASCII module sets the data written by the EX CPU to its internal WRITE buffer memory.

(2) After transferring all data, the BASIC-52 program No. is transferred to the ASCII module requesting to activate the program of this No.

(3) The ASCII module executes the BASIC-52 program and processes the data stored in the WRITE buffer memory.

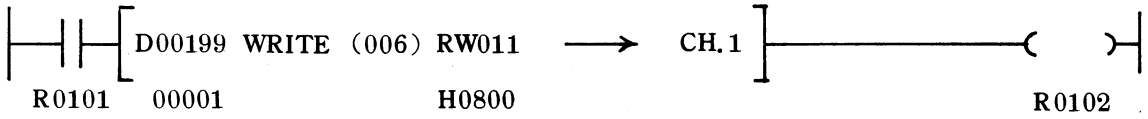
(4) The execution of the WRITE command is finished when the ASCII module finishes executing the BASIC-52 program.

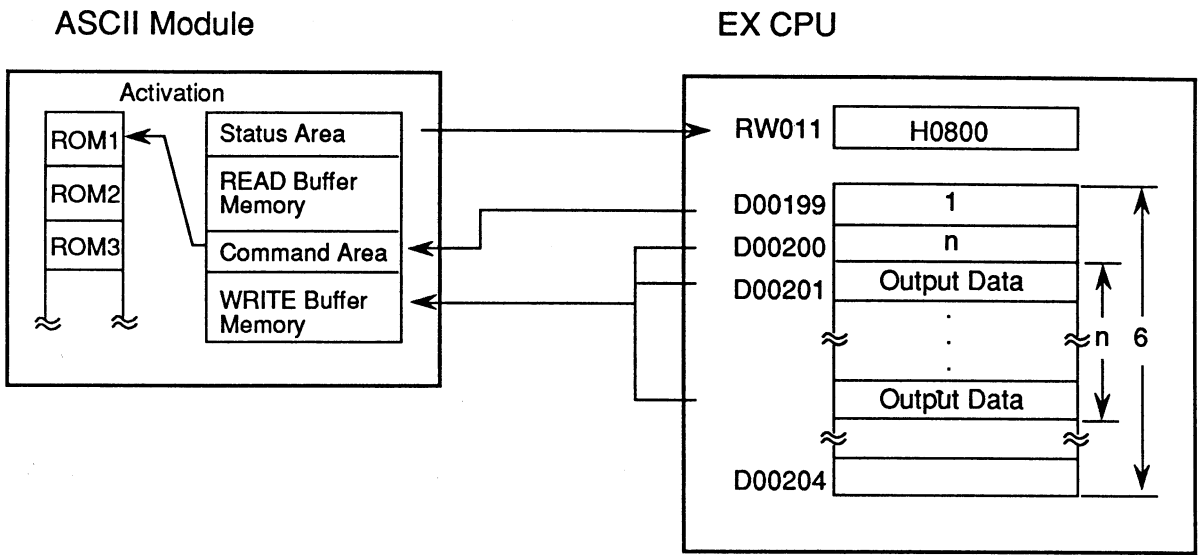
Refer to page 57 and after for the details of the Completion Status Register.



WRITE Command Timing Chart

Program Example



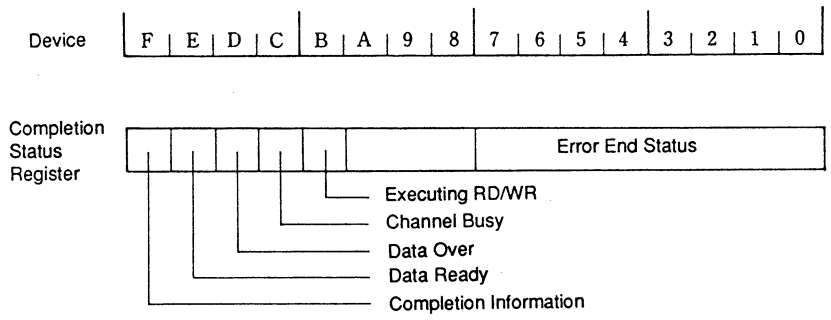


- Outputs data stored in the table headed by D00201 of the size shown by the data of register D00200 with the ASCII module shown by CH.1 when normally open contact R0101 is ON.
- Executes the BASIC-52 program of the No. designated by the data of D00199.
- The completion status will be shown in RW011.

Note: • Maintain the condition input while the ASCII module is executing processing.

• The output data size n will also be transferred to the ASCII module.

The READ/WRITE commands indicate the status of the EX CPU and ASCII module during the execution of these commands, as well as processing results after command execution and other data, in the completion status register as illustrated below.



The devices are described below.

Device	Name	Function
F	COMPLE-TION INFORMA-TION	This is the flag to show status after completing execution of READ/WRITE commands. If this flag is '0', the READ/WRITE commands have ended normally. If '1', an error has occurred and the command execution ended in error. The details of error ending are stored as codes in lower eight devices of 0 to 7.
E	DATA READY	This flag shows whether data transfer from the READ DATA buffer of the ASCII module to the input data area has ended.
D	DATA OVER	This device is set to '1' if READ data from the ASCII module surpasses the maximum table size set by the EX CPU. The EX CPU reads data up to the maximum table size, and it discard data that cannot be read.
C	CHANNEL BUSY	This device is set to '1', and an error results, if the READ/WRITE commands are activated with an ASCII module that is already activated by the READ/WRITE commands and is executing a

Device	Name	Function
C	CHANNEL BUSY	program that uses several READ/WRITE commands. By keeping the condition input on as it is, the busy status will be released and the command will be executed as soon as the READ/WRITE commands being executed end.
B	EXECUTING RD/WR	READ/WRITE commands are being executed while this device is set to '1'.
7 | 0	ERROR END STATUS	If READ/WRITE commands result in an abnormal end, the detailed error code is stored. Refer to the following error code list for the details of individual error codes.

List of Error Codes

Error Code	Error	Description of Error
01H	NO MODULE ERROR	This error code is set if an ASCII module is not mounted in the designated channel number.
02H	CHANNEL H/W ERROR	This error code is set if a hardware error occurs in the ASCII module of the designated channel number and access by the EX CPU is impossible. Once this error occurs, this error code is set if READ/WRITE commands are activated with this ASCII module.
03H	CHANNEL NOT IDLE ERR	This error code is set if the ASCII module of the designated channel number is not in an IDLE state (so that commands from EX CPU can be accepted)during the execution of the READ/WRITE command. For example, this error results if an ASCII module is suddenly reset or if the EDIT mode of a BASIC-52 program is suddenly set.

Error Code	Error	Description of Error
04H	ABNORMAL I/O RESPONSE ERROR	This error code is set if an I/O NO SYNCHRO error occurs during execution of the READ/WRITE commands by EX CPU.
05H	CHANNEL PROCESSING MODE CHANGE ERROR	This error code is set if the ASCII module is put to an IDLE state before processing of READ/WRITE commands is finished.
06H	CHANNEL CPU ERROR	This error code is set if the ASCII module of the designated channel number detects a CPU error by self-diagnosis function. The CPU error means that the BASIC-52 utility routine or BASIC-52 interpreter of ASCII module does not operate properly.
07H	CHANNEL RAM ERROR	This error code is set if the ASCII module of the designated channel number detects a RAM error by self-diagnosis function. The RAM error indicates that data cannot be written or read normally with the data memory (RAM) of the ASCII module.
08H	CHANNEL ROM ERROR	This error code is set if a ROM error is detected by the ASCII module of the designated channel number by self-diagnosis function. The ROM error indicates that system data of the ASCII module cannot be read properly.
09H	CHANNEL WD TIMER ERROR	This error code is set if the ASCII module of the designated channel number generates a watchdog timer error by the watchdog timer set by the user. Refer to the description of special function IE in APPENDIX 5. Description of BASIC-52 Language for additional information. The ASCII module is automatically reset after it causes watchdog timer error.

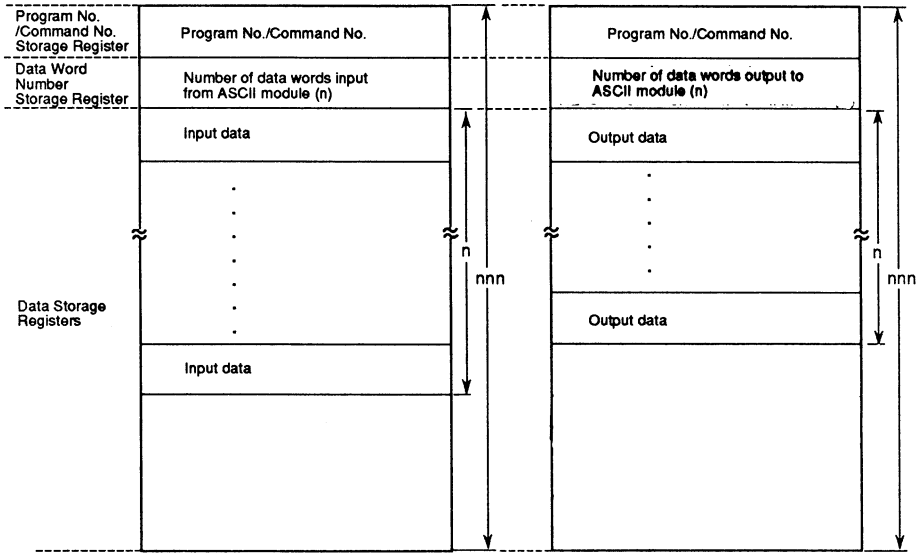
Error Code	Error	Description of Error
0AH	CHANNEL PROGRAM NO. ERROR	This error code is set if the number of the BASIC-52 program sent to the ASCII module of the designated channel number is illegal. (Program number must be an integer "n" where 0 < n ≤ [number of existing programs].)
0BH	CHANNEL ILLEGAL COMMAND ERROR	This error code is set if an illegal command is sent to the ASCII module of the designated channel number, (command No. does not exist).
0CH	CHANNEL INITIALI-ZATION ERROR	This error code is set if the ASCII module of the channel number designated by the EX CPU is executing an initialization routine, when it is activated by a READ/WRITE command.
82 H	INVALID TABLE SIZE ERROR	This error code is set if the set table size is not within the effective range. The effective table size range is as follows: EX2000: 2 ~ 502 EX 250/500: 2 ~ 252
83H	ABNORMAL REGISTER RANGE ERROR	This error code is set if the set table size surpasses the register area.
84H	ABNORMAL PROGRAM NO. RANGE ERROR	This error code is set if the number of the BASIC-52 program sent to the ASCII module of the designated channel number is outside of the effective range. (Program number must be an integer "n" where 0 < n ≤ [number of existing programs].
85H	PROGRAM NO. MISMATCH ERROR	This error code is set if the program numbers designated in READ and WRITE commands during simultaneous execution of READ/WRITE command are not identical. Refer to 6.5 READ/WRITE Simultaneous Execution Processing of CHAPTER 6. PROGRAMMING APPLICATION for the details

Error Code	Error	Description of Error
		of READ/WRITE simultaneous execution.
86H	WRITE DATA SIZE ERROR	This error code is set if the number of words set surpasses the effective range inside the table size during WRITE processing.

The meanings of the parameters are described below.

Parameters are stored in the table whose size is decided by [nnn] during the execution of a READ/WRITE command. The parameter table configuration is shown below.

<During READ Command> <During WRITE Command>



The effective value ranges of the registers in the parameter table are shown below.

Register in Parameter Table	Effective Value Range
Program No./command No. storage register	1 ~ 511 (Program Nos. 1 to 255. Command Nos. 256 to 511.)
Data word number storage register	0 ~ 500
Data storage resisters	Range of registers that can be handled by EX CPU.

An error end results if a value outside of the effective range is set and a READ/WRITE command is executed.

The effective value ranges of the resisters to store matching program numbers /command numbers are:

Program Numbers: 1-255 (BASIC-52 program numbers stored in the EEPROM inside the ASCII module)

Command Numbers: 256-511 (Command numbers sent by the EX CPU to the ASCII module)

The command number and matching processing are shown below.

Command No.	Processing
261	Requests ASCII module to reset.

Numbers 256 to 260 and 262 to 511 are reserved.

5.3 BASIC-52 Programs on ASCII Module Side

This section describes the BASIC-52 programs on the ASCII module side during basic READ and WRITE operations. This section outlines the items related to interfacing with the EX CPU. Refer to APPENDIX 5. DESCRIPTION OF BASIC-52 LANGUAGE for the individual BASIC-52 commands. The BASIC-52 programs on the ASCII module side for READ and WRITE processing use several CALL subroutines. PUSH, POP, and other statements are also used. Processing of these is provided specially to support interfacing between the EX CPU and ASCII module.

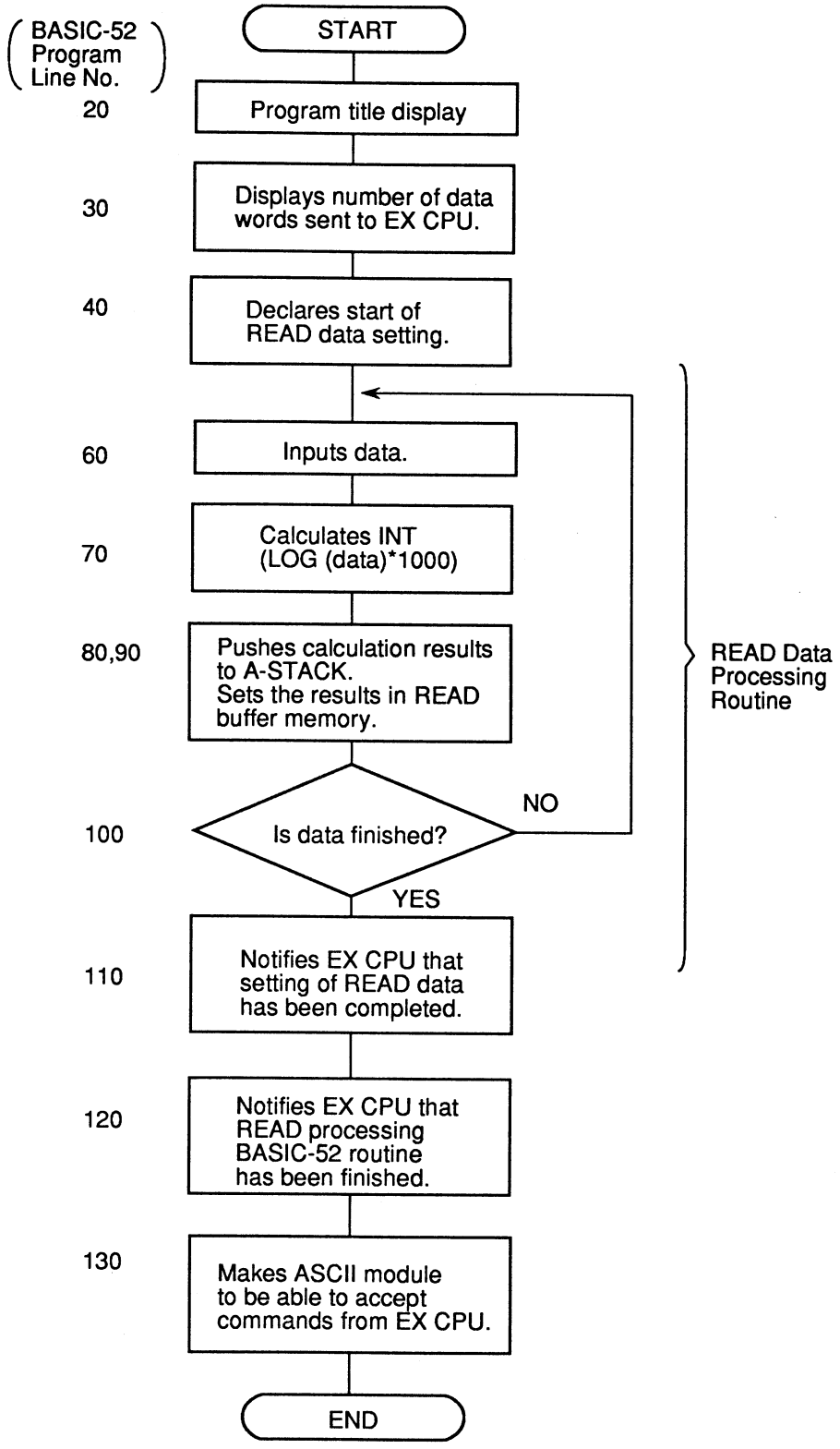
CALL routines can be divided roughly into two types. The first type relates to data exchange between the EX CPU and ASCII module. The second type relates to flag processing of the ASCII module prior to interfacing between the EX CPU and ASCII module.

The functions of these CALL subroutines are described below.

5.3.1 READ processing BASIC-52 programs

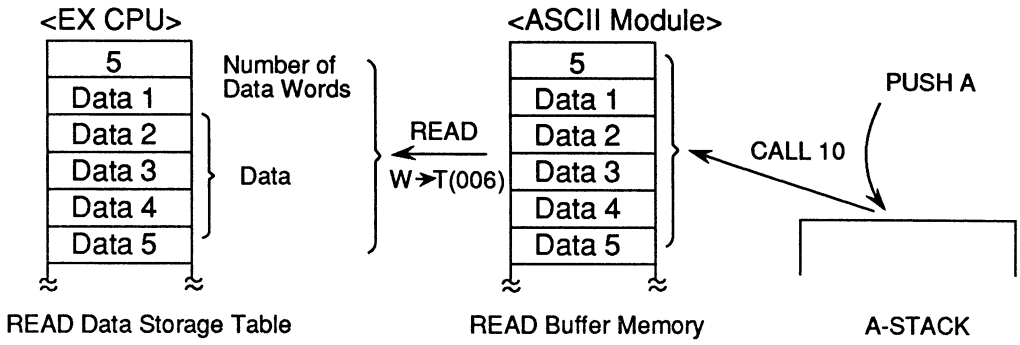
This section describes the BASIC-52 programs for basic READ processing. As an example, the following BASIC-52 program will be discussed:

```
10      REM SAMPLE READ
20      PRINT "SAMPLE READ"
30      DIM A(10)
40      PRINT "DATA TO SEND TO EX =5"
50      FOR I=1 TO 5
60      INPUT "DATA ? ",B
70    A(I)=INT(LOG(B)*1000)
80      NEXT I
90      CALL 19
100     FOR J=1 TO 5
110     PUSH A(J)
120     CALL 10
130     NEXT J
140     CALL 20
150     CALL 13
160     CALL 16
```


BASIC-52 Program Line No.
START
20
Program title display
30
Displays number of data words sent to EX CPU.
40
Declares start of READ data setting.
60
Inputs data.
70
Calculates INT (LOG (data)*1000)
80,90
Pushes calculation results to A-STACK. Sets the results in READ buffer memory.
100
Is data finished?
NO
YES
110
Notifies EX CPU that setting of READ data has been completed.
120
Notifies EX CPU that READ processing BASIC-52 routine has been finished.
130
Makes ASCII module to be able to accept commands from EX CPU.
END
READ Data Processing Routine

When the CALL subroutines used in the BASIC-52 programs shown in page 63 are classified, CALL 10 can be classified as related to data exchange, while CALL 19, 20, 13, and 16 can be classified as related to flag processing.

CALL 10 processing is outlined below.



The data handling formats of the EX CPU and ASCII module sides differ. Therefore, the ASCII module pushes data to the argument stack (A-STACK) using a PUSH statement. Using a CALL 10 subroutine, data type conversion processing is then performed to set data in the READ buffer memory. The EX CPU reads data in the READ buffer memory using the DEMULTIPLEXER or READ command and transfers data to the table. The diagram above illustrates this processing.

The subroutines CALL 19, 20, 13, and 16 are processed roughly as follows:

CALL 19: Declares during the execution of a BASIC-52 program that data setting in the READ buffer memory will be started.

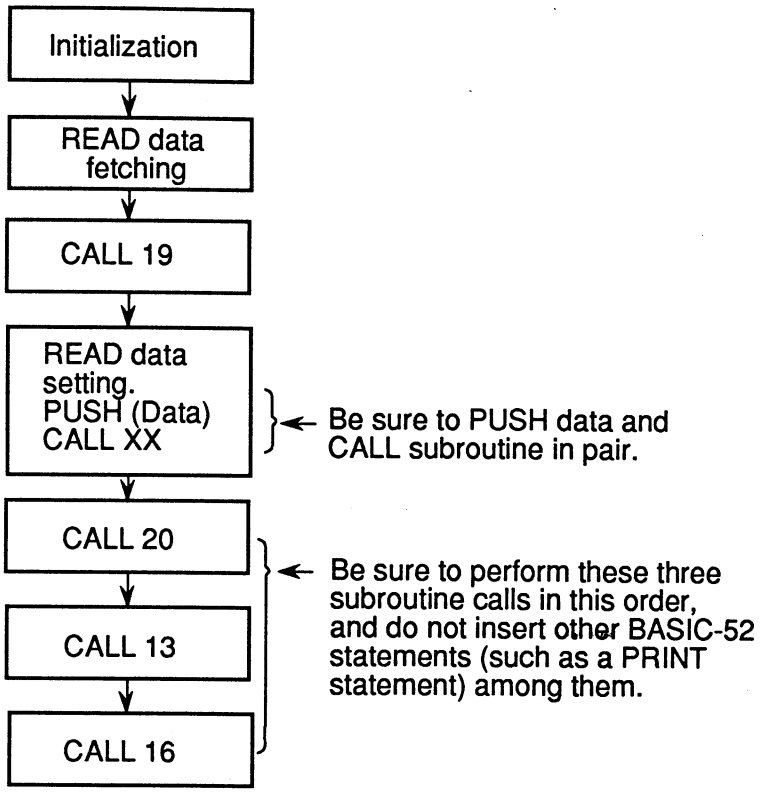
CALL 20: Notifies the EX CPU that setting of READ data has been finished. (READ data set end device is set to 1.)

CALL 13: Notifies the EX CPU that the BASIC-52 routine for REAd processing has been finished. (READ data set end device is reset and READ acknowledge enable device is reset to 1.)

CALL 16: Initializes so that commands from the EX CPU can be accepted again.

Refer to the explanations given in parentheses and to section 5.1.2 Basic READ Operation if the ASCII module is used as an I/O module.

The operating method of the CALL subroutines in the BASIC-52 programs for READ processing and the timing to use them are described below.



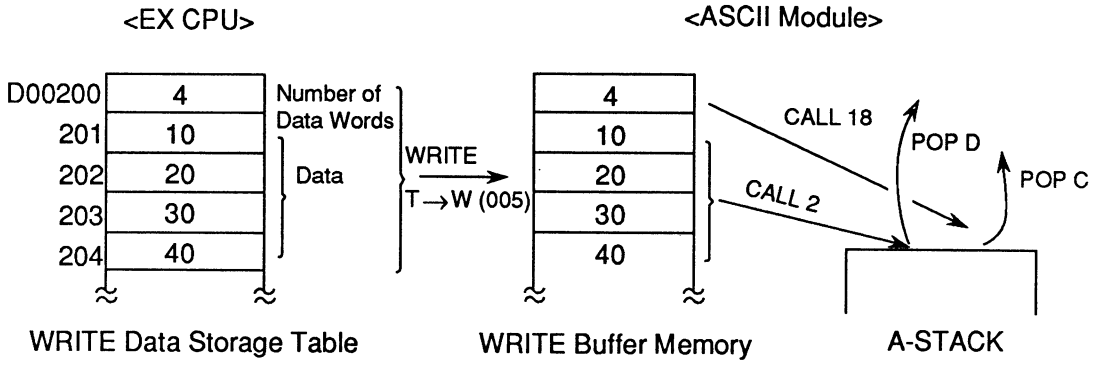
5.3.2 WRITE processing BASIC-52 programs

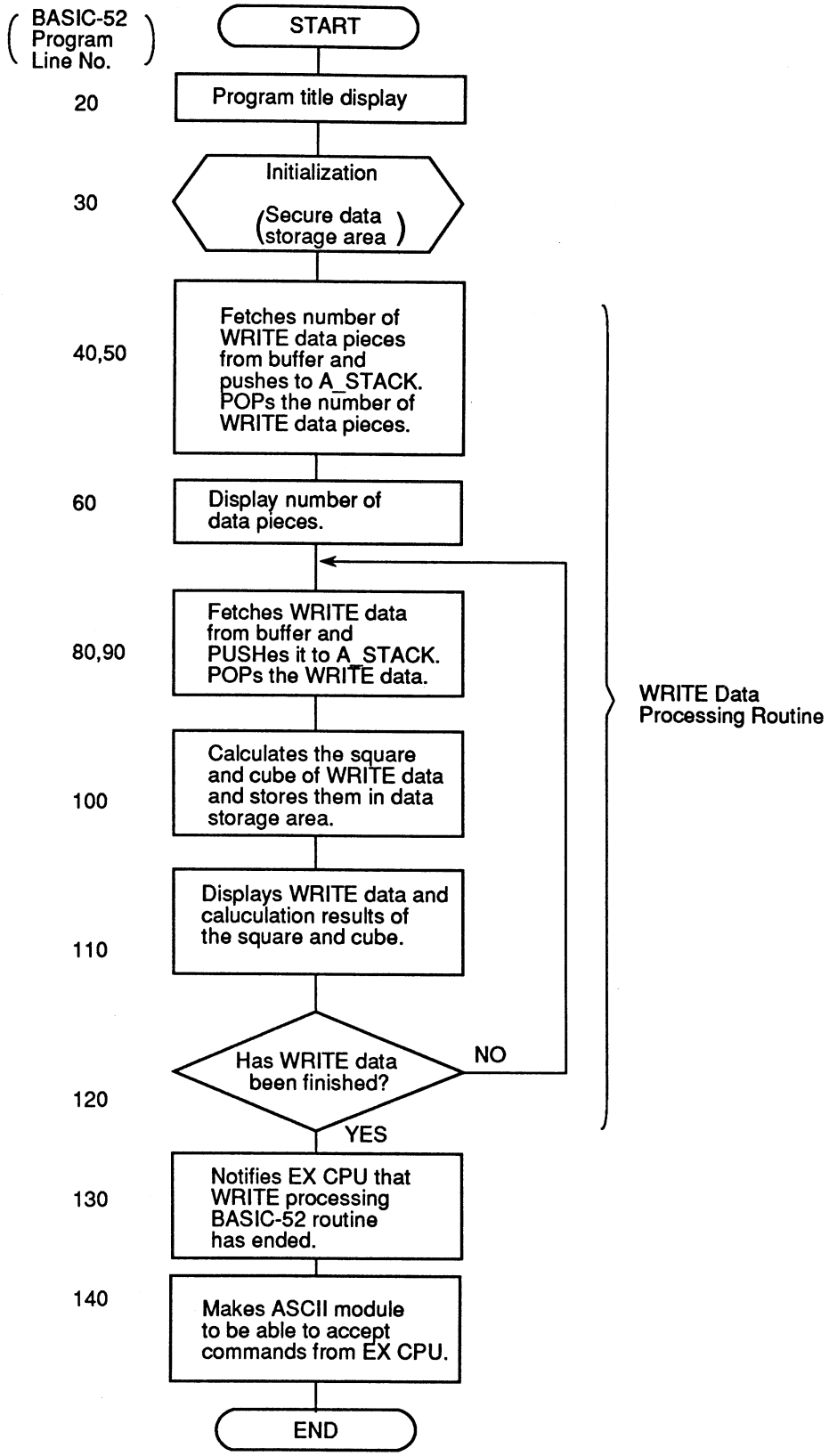
This paragraph describes the BASIC-52 programs for WRITE processing. As an example, the sample BASIC-52 program used in chapter 4 will be discussed. The sample program is as follows:

```
10      REM SAMPLE WRITE
20      PRINT "SAMPLE WRITE"
30      DIM A(10), B(10)
40      CALL 18
50      POP C
60      PRINT "DATA SENT =",C
70      FOR I=1 TO C
80      CALL 2
90      POP D
100   A(I)=D**2 : B(I)=D**3
110     PRINT USING(######),D,A(I),B(I)
120     NEXT I
130     CALL 14
140     CALL 16
```

When the CALL subroutines used in the above BASIC-52 program are classified, CALL 18 and 2 can be classified as related to data exchange, and CALL 14 and 16, as related to flag processing.

CALI 18 and 2 are processed roughly as follows:




BASIC-52 Program Line No.
START
20
Program title display
30
Initialization
(Secure data storage area)
40,50
Fetches number of WRITE data pieces from buffer and pushes to A_STACK. POPs the number of WRITE data pieces.
60
Display number of data pieces.
80,90
Fetches WRITE data from buffer and PUSHes it to A_STACK. POPs the WRITE data.
100
Calculates the square and cube of WRITE data and stores them in data storage area.
110
Displays WRITE data and caluculation results of the square and cube.
120
Has WRITE data been finished?
NO
YES
130
Notifies EX CPU that WRITE processing BASIC-52 routine has ended.
140
Makes ASCII module to be able to accept commands from EX CPU.
END
WRITE Data Processing Routine

Using the MULTIPLEXER or WRITE command, the EX CPU transfers the data stored in the table to the WRITE buffer memory inside the ASCII module. However, the data storage format of the EX CPU and the format of the ASCII module side differ. For this reason, using the CALL 18 and 2 routines, the ASCII module converts the format of the data received from the EX CPU and PUSHes the data to the argument stack (A_STACK). The ASCII module fetches and uses this data using a POP statement during the execution of a BASIC-52 program.

The diagram in page 67 illustrates this processing.

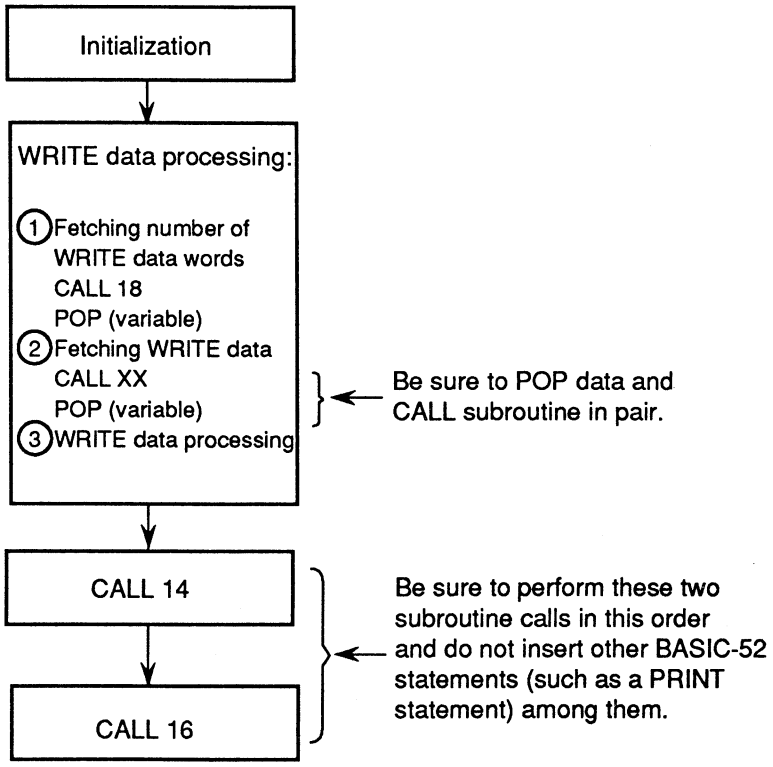
The CALL 14 and 16 subroutines are processed roughly as follows.

CALL 14: Notifies the EX CPU that the execution of a WRITE processing BASIC-52 routine has been finished. (Resets the WRITE Acknowledge Enable Device to "1.")

CALI 16: Initializes so that commands from the EX CPU can be acknowledged again.

Refer to the explanations given in parentheses and to section 5.1.3 Basic WRITE Operation if the ASCII module is used as an I/O module.

The operating method of the CALL subroutines in the BASIC-52 programs for WRITE processing and the timing to use them are described below.



This chapter describes the application functions of the ASCII module using specific examples.

6.1 READ Processing

READ processing performs the processing described in section 5.1.2. Basic READ Operation and in the description of the READ commands in section 5.2.2 READ/WRITE Commands, as outlined below.

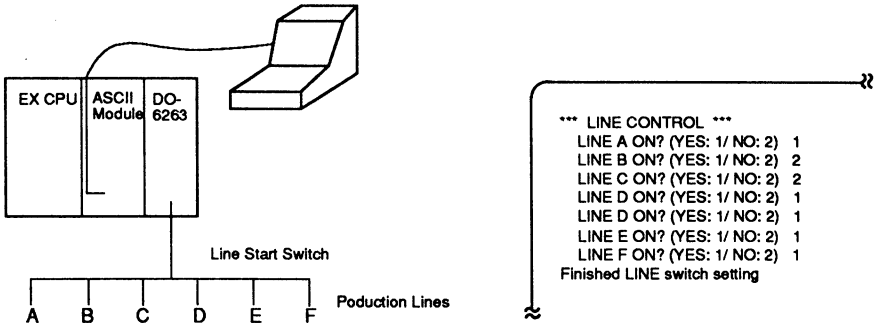
(1) Using the READ sequence and READ command, the EX CPU requests the ASCII module to start a specified BASIC-52 program.

(2) The ASCII module fetches data from the keyboard of the console device, bar code reader or other device, during a BASIC-52 program activated as mentioned in (1). The ASCII module sets the data in the READ buffer memory to enable the EX CPU to read it.

(3) The EX CPU reads the set data.
This ends one cycle of processing.

[Program Example]

The console device connected to CH1 of the ASCII module displays a message when the EX CPU starts a program. The message prompts input to decide whether or not to switch on production lines A to F. The data input by the operator is read by the EX CPU, which outputs the production line ON/OFF information through the DC output module (DO-6263).



An example of the program on the EX CPU side when the ASCII module is used as an I/O module is shown below. I/O are assigned in this case as follows:

Slot No.	I/O Type	Register No.	Assignment	
0	X+Y 04W	XW 000	ASCII module	Status register
		XW 001		READ data register
		YW 002		Command register
		YW 003		WRITE data register
1	Y 01W	YW 004	DO-6263	

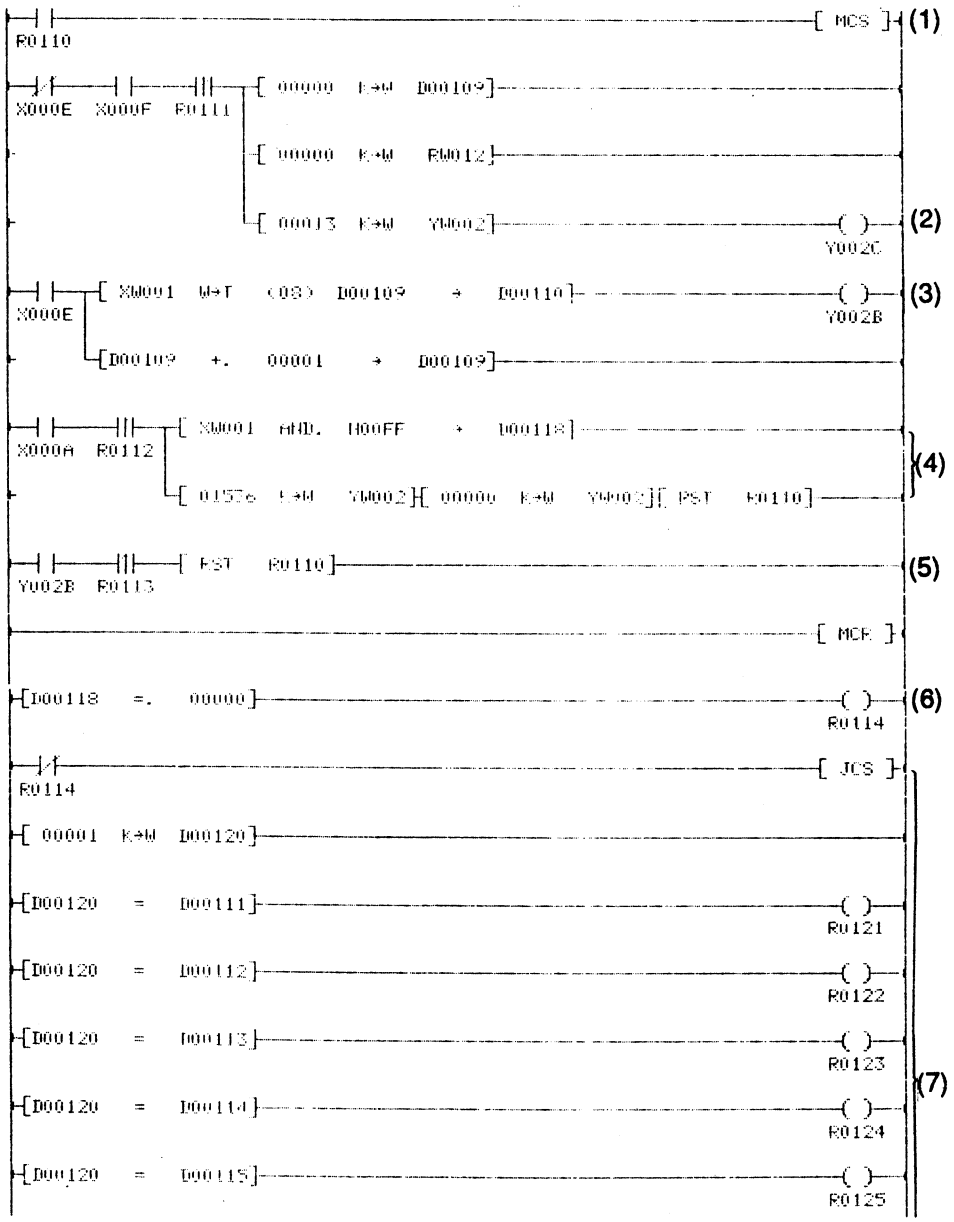
This sequence program activates the ROM13 program on the ASCII module side.

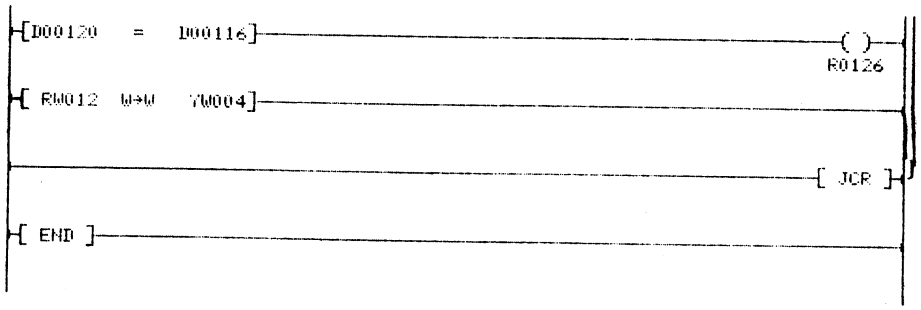
The functions of the devices and registers used in this program are shown below. Descriptions of the functions of the status registers and command registers of the ASCII module are omitted.

Device or Register	Function
R0110	READ process activation
R0111	Differential contact to execute the command for one scan time only to transfer BASIC-52 program No. and to switch on the READ start device of the ASCII module command register.
R0112	Differential contact to execute error code reading and ASCII module resetting only one scan time if an error occurs.
R0113	Differential contact to execute the command to reset the READ processing start device only one scan time.
R0114	The device to decide whether or not to set the production line start switch.
RW012	Register to store ON/OFF information of the production line start switch
D00109	Pointer for input data (READ data) storage area.
D00110	Register to store the number of words of input data (READ data) [7 words].
D00111 to D00117	Area to store input data (READ data).

Device or Register	Function
D00118	Register to store error codes to be set by the ASCII module if an error occurs.
D00120	Register to store data for comparison to set switch ON/OFF information.

[READ Sequence Program]





(1) Starts the READ sequence processing if R0110 switches on.

(2) If the READ data set end device is '0' and READ acknowledge enable device is '1', sends the program No. of the BASIC-52 program to be started to the ASCII module, then sets the READ START device.

(3) If the READ data set end device becomes '1', reads the READ data, which is set by the ASCII module, in registers after data register D00110. Sets the data READ end device after reading.

(4) The error code is stored in D00118 if an error occurs. The WRITE start device and WRITE data set end device are set simultaneously to reset the ASCII module and to end the READ sequence processing.

(5) If the READ END device becomes '1', ends the READ sequence processing.

(6) Performs processing between JCS and JCR if correct data is input (unless error code is set in D00118).

(7) Sends line switch ON/OFF information to DO-6263 based on data read in D00111 to D00116.

Program examples on the EX CPU side when the ASCII module is used as an option module are shown below. I/O assignment as this time will be as follows:

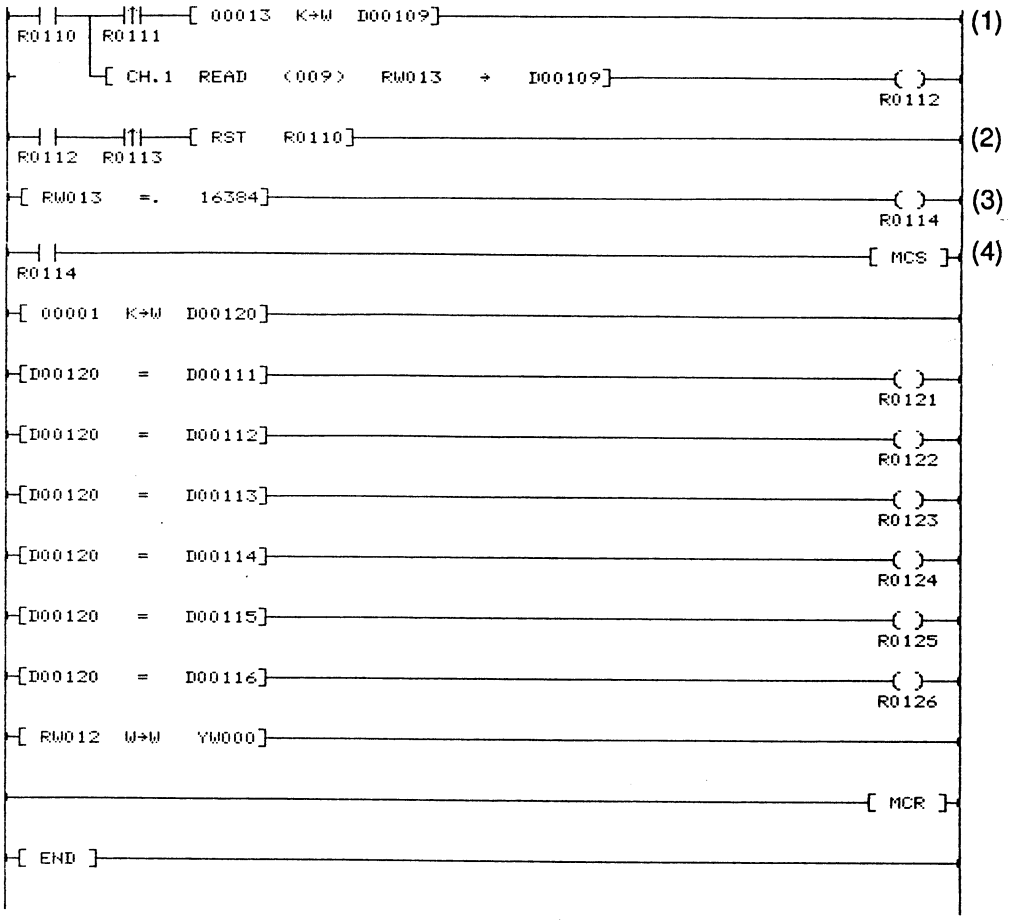
Slot No.	I/O Type	Register No.	Assignment
0	OPT	———	ASCII module
1	Y 01W	YW 000	DO-6263

This sequence program activates the ROM13 program on the ASCII Module.

The functions of the devices and registers used in this program are shown below.

Device or Register	Function
R0110	READ processing start
R0111	Differential contact to execute the command to set BASIC-52 program No. only 1 scan time
R0112	READ command end output
R0114	Device to decide whether or not to set production line start switch
RW012	Register to store ON/OFF information of production line start switch
RW013	READ command completion status register
D00109	Register to store BASIIC-52 program No.
D00110	Register to store number of input data (READ data) words [7 words]
D00111 to D00117	Area to store input data (READ data)
D00120	Register to store data for comparison to set switch ON/OFF information

[READ Command Program]



(1) Starts the READ command processing if R0110 switches on.

(2) Ends READ command processing after reading data.

(3) Checks the value of the completion status register, and switches on R0114 if an error has not occurred.

(4) If READ command processing ends normally, sends line switch ON/OFF information to DO-6263 based on the data read in D00111 to D00116.

An example of a BASIC-52 program on the ASCII module side for READ processing is shown.

[READ Processing BASIC-52 Program]

```
10  REM *** SINGLE READ SAMPLE PROGRAM ***
20  PRINT "*** LINE CONTROL ***"
30  STRING 100,10
40  FOR I=0 TO 5
50   PRINT TAB(5),"LINE ",CHR(41H+I), : INPUT" ON? (YES:1/NO:2) ",B
60   IF B=1 THEN A(I)=B : GOTO 80
70   IF B=2 THEN A(I)=0 ELSE GOTO 50
80  NEXT I
90  CALL 19
100 FOR I=0 TO 5
110  PUSH A(I)
120  CALL 10
130 NEXT I
140 $(0)="E"
150 CALL 8
160 PRINT "FINISHED LINE SWITCH SETTING."
170 CALL 20
180 CALL 13
190 CALL 16
```

(1) (2) (3) (4) (5) (6)

(1) Sets ON/OFF information of the start switches of production lines A to F in B (0) to B (5).

(2) Declares start of READ data setting.

(3) Sets READ data in the buffer memory.

(4) Notifies the EX CPU that READ data setting has been completed.

(5) Notifies the EX CPU that single READ processing has ended.

CAUTION

Be sure to perform CALL16 in succesion afterward.

(6) Sets to the mode to queue up for command input by the EX CPU.

6.2 WRITE Processing

WRITE processsing performs processing explained in paragraphs 5-1-3 Basic WRITE Operation and 5-2-2 READ/WRITE Commands which describe the WRITE command. The outline of it is given below.

(1) Using the WRITE sequence and WRITE command, the EX CPU writes data in the ASCII module.

(2) The ASCII module processes this data in the BASIC-52 program designated by the WRITE sequence/WRITE command. This ends one cycle of processing.

[Program Example]

The EX CPU fetches data from the analog equipment connected to the analog input module (AI-6290) and temporarily stores it in the data registers. The EX CPU writes the data in the ASCII module when the number of data pieces fetched becomes 500. The BASIC-52 program of the ASCII module converts values on the console device.

An example of program of the EX CPU side when te ASCII module is used as an I/O module is shown below.

The I/O assignment is shown below.

Slot No.	I/O Type	Register No.	Assignment	
0	X+Y 04W	XW 000	ASCII module	Status register
		XW 001		READ data register
		YW 002		Command register
		YW 003		WRITE data register
1	X 02W	XW 004	AI-6290	
		XW 005		

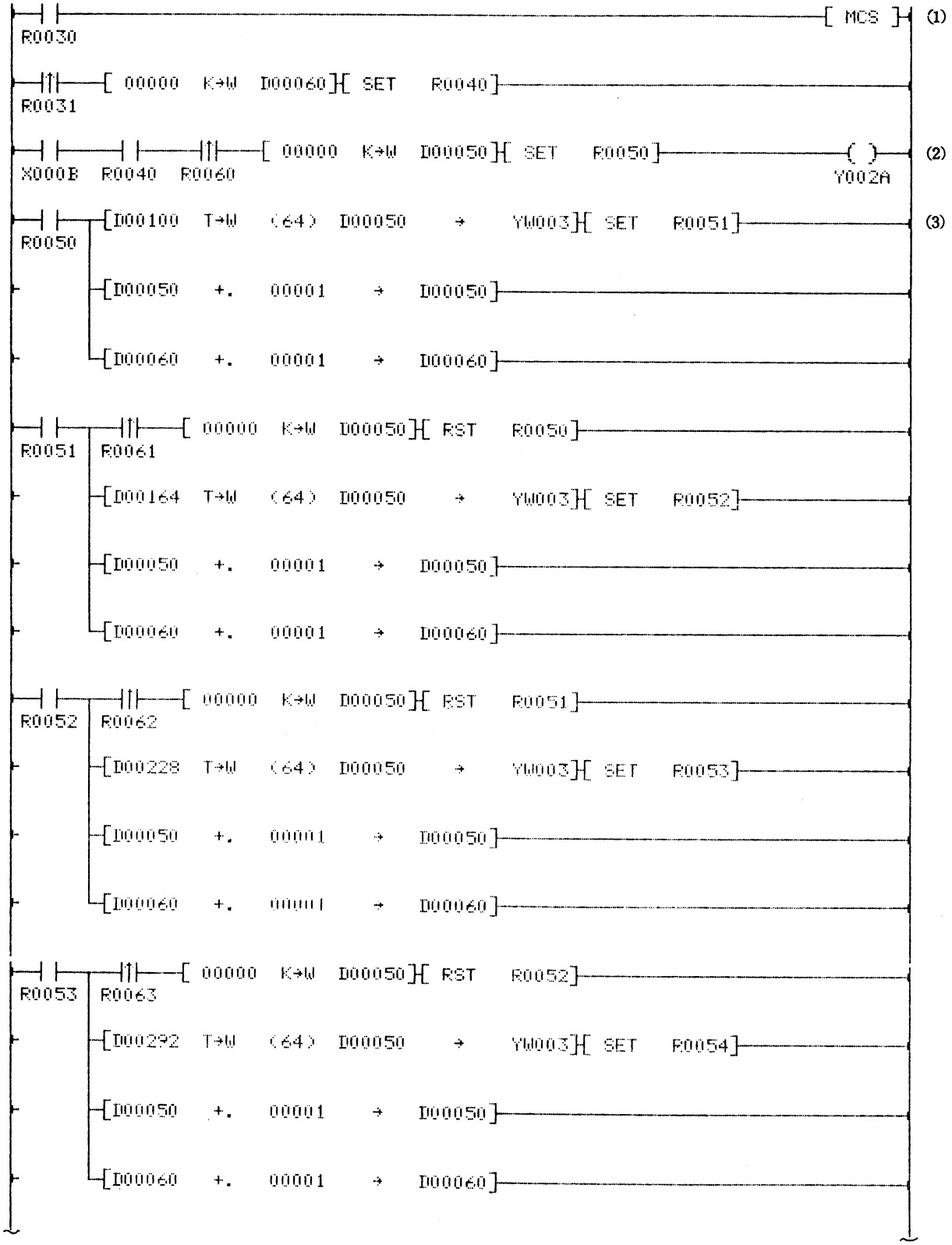
This sequence program also activates the ROM14 program on the ASCII module side.

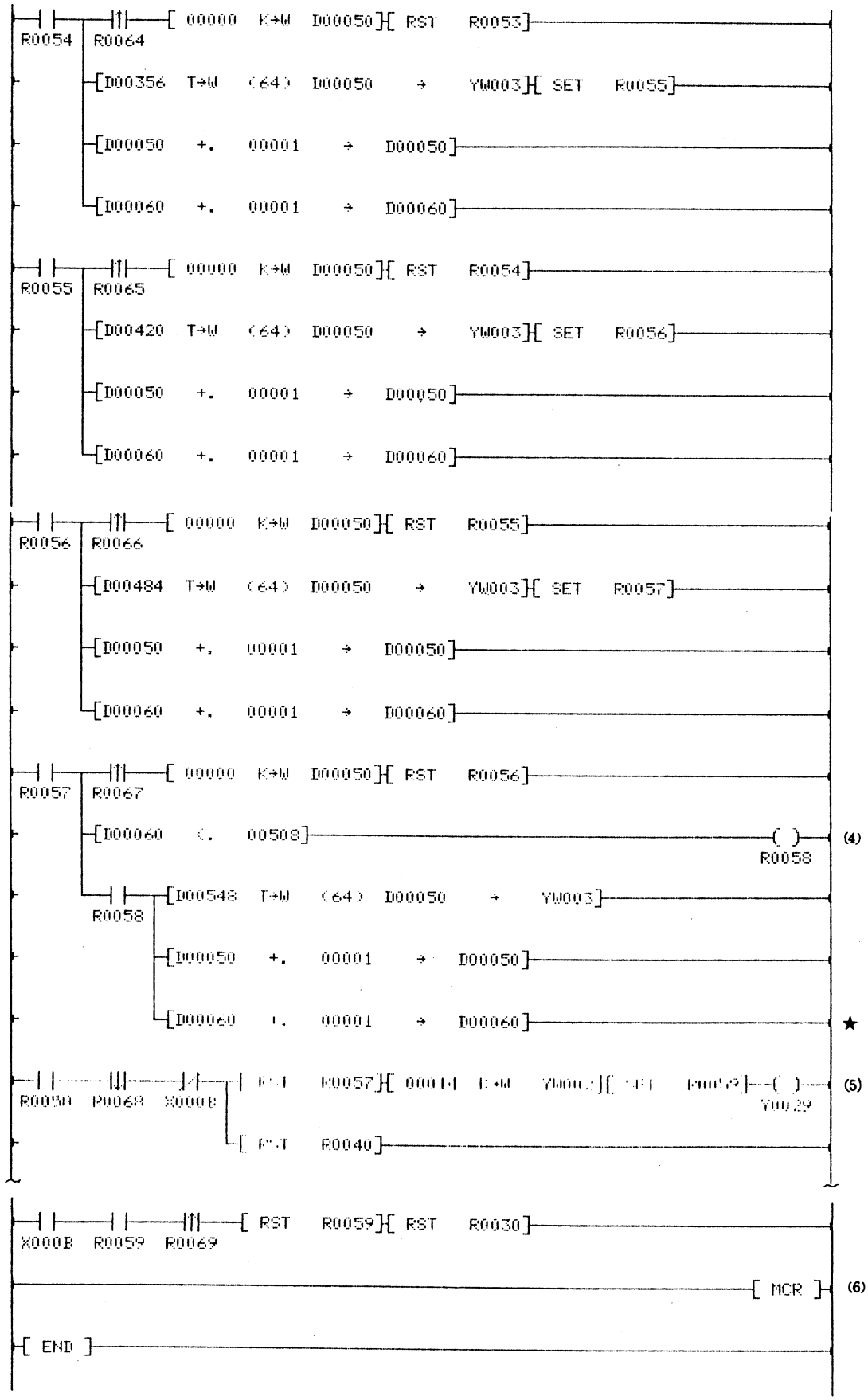
The functions of the devices and registers used in this program are shown below.

Device or Register	Function
R0030	WRITE processing start.
R0031	Differential contact to execute the command to clear analog data transfer counter and to set device 'analog data being transferred' only one scan time.
R0040	Analog data being transferred.
R0050 R0051 R0052 R0053 R0054 R0055 R0056 R0057	The devices to execute the command to transfer analog data stored from D00101 to D00600 and the number of output data words stored in D00100 to the ASCII module. R0050 is ON: Data of from D00100 to D00163 is transferred. R0051 is ON: Data of from D00164 to D00227 is transferred. R0052 is ON: Data of from D00228 to D00291 is transferred. R0053 is ON: Data of from D00292 to D00355 is transferred. R0054 is ON: Data of from D00356 to D00419 is transferred. R0055 is ON: Data of from D00420 to D00483 is transferred. R0056 is ON: Data of from D00484 to D00547 is transferred. R0057 is ON: Data of from D00548 to D00600 is transferred.
R0058	Interlock to prevent writing data after D00601 in the ASCII module.
R0059	BASIC-52 program being executed.
R0060 R0061 R0062 R0063 R0064 R0065 R0066 R0067	Differential contact to execute the command to clear the analog data storage area pointer and to reset condition input of the command, for which data transfer has been finished, only 1 scan time.
R0068	Differential contact to execute command to transfer BASIC-52 program No. and to set the WRITE data set end device for the ASCII module command register only 1 scan time.
R0069	Differential contact to execute the command to reset the devices for BASIC-52 program being executed and for WRITE processing start only one scan time.

Device or Register	Function
D0050	Pointer for analog data stored from D00101 to D00600.
D0060	Analog data transfer counter.
D00100	Register to store number of output data (WRITE data) words (500 words).
From D00101 to D00600	Analog data storage area.

[WRITE Sequence Program]




R0054
R0064
00000 K→W D00050
RST R0053
D00356 T→W (64) D00050 → YW003
SET R0055
D00050 +. 00001 → D00050
D00060 +. 00001 → D00060
R0055
R0065
00000 K→W D00050
RST R0054
D00420 T→W (64) D00050 → YW003
SET R0056
D00050 +. 00001 → D00050
D00060 +. 00001 → D00060
R0056
R0066
00000 K→W D00050
RST R0055
D00484 T→W (64) D00050 → YW003
SET R0057
D00050 +. 00001 → D00050
D00060 +. 00001 → D00060
R0057
R0067
00000 K→W D00050
RST R0056
D00060 <. 00508
R0058
(4)
R0058
D00548 T→W (64) D00050 → YW003
D00050 +. 00001 → D00050
D00060 +. 00001 → D00060
★
R0059
R0069
X000B
RST R0057
00014 K→W YW000
SET R0059
Y00029
(5)
RST R0040
X000B R0059 R0069
RST R0059
RST R0030
MCR
(6)
END

(1) Starts WRITE sequence processing if R0030 switches on.

(2) Sets the WRITE START device if the WRITE acknowledge enable device is '0.'

(3) Processing between here and the location marked '★' writes the number of data words stored in data register D00100 and 500-word data stored in data registers D00101 to D00600 to the ASCII module.

(4) Arrange a limit check in the sequence program so that no more than 501 words of data is written if the WRITE sequence is used.

(5) Sends the No. of BASIC-52 program to be started to the ASCII module after writing all data , and sets the WRITE data set end device.

(6) Ends WRITE sequence processing when the ASCII module ends WRITE data processing and the WRITE acknowledge enable device again becomes '1.'

A program example on the EX CPU side when the ASCII module is used as an option module is shown below. The I/O assignment at this time will be as follows:

Slot No.	I/O Type	Register No.	Assignment
0	OPT	———	ASCII module
1	X 02W	XW 000	AI-6290
		XW 001	

This sequence program activates the ROM14 program on the ASCII Module side.

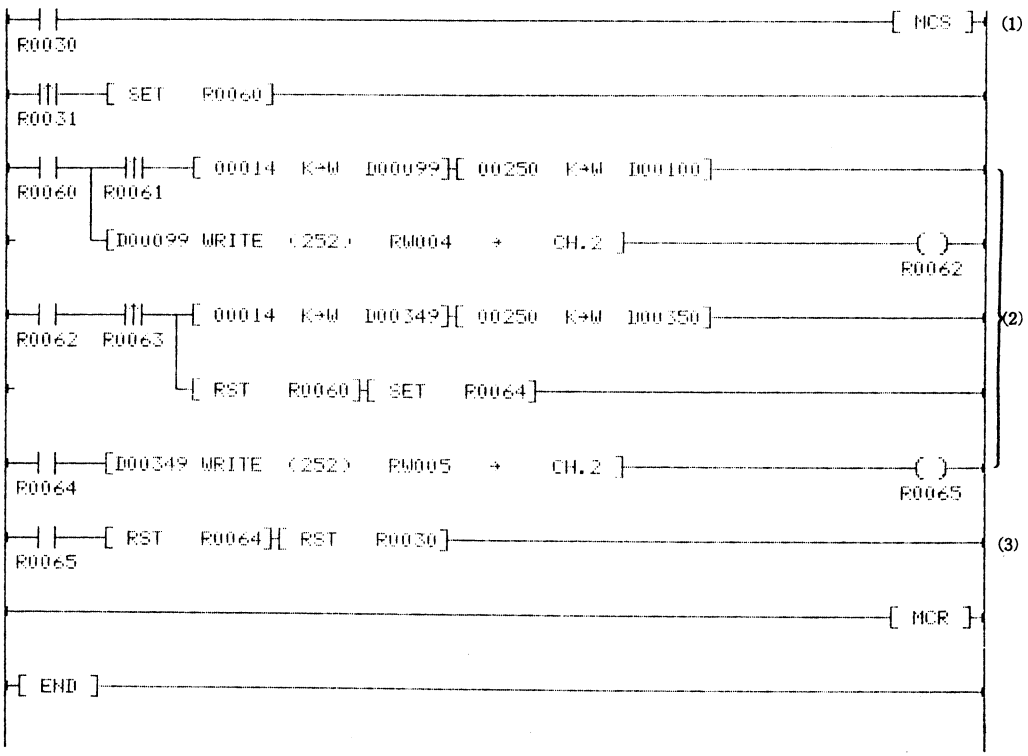
The programming methods differ slightly when EX250 or EX500 is used and when EX2000 is used as the EX CPU. A program example when EX250 or EX500 is used is presented first.

The functions of the devices and registers used in the program are shown below.

Device or Register	Function
R0030	WRITE processing start.
R0031	Differential contact to execute the command to set first WRITE command start device only one scan time.
R0060	First WRITE command start.
R0061	Differential contact to execute command to set BASIC-52 program No. and number of transfer words only one scan time by first WRITE command.
R0062	End output of first WRITE command.
R0063	Differential contact to execute the command to set BASIC-52 program No. and number of transfer words only one scan time by second WRITE command.
R0064	Second WRITE command start.
R0065	End output of second WRITE command.
RW004	Completion status register of first WRITE command.

Device or Register	Function
RW005	Completion status register of second WRITE command.
D00099	Register to store BASIC-52 program No. in first WRITE command.
D00100	Register to store number of output words (250 words) in first WRITE command.
D00101 to D00600	Analog data storage area.

[WRITE Command Program (EX250/500)]



(1) Starts WRITE command processing when R0030 switches on.

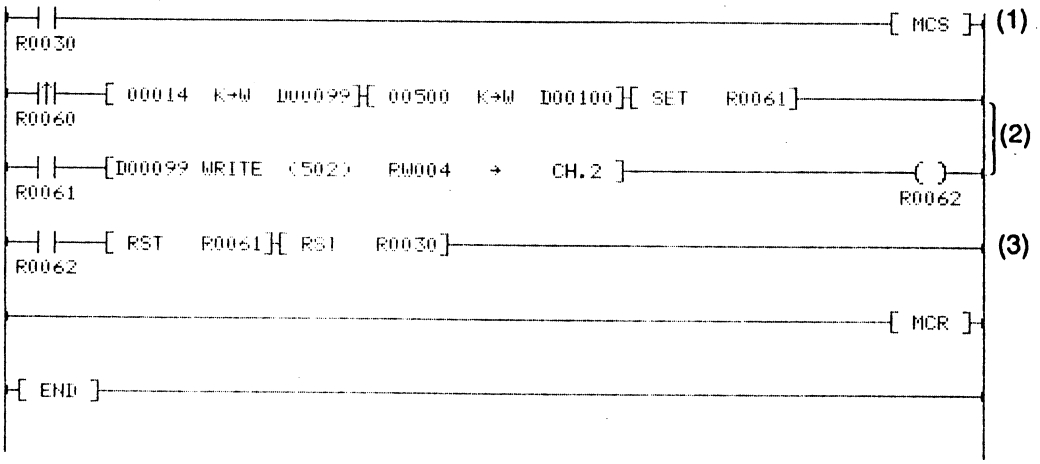
(2) Maximum data sent by EX250/500 using the WRITE command at one time is 250 words. Data of 500 words is sent by dividing it into two as shown here.

(3) Ends WRITE command processing when the execution of the two WRITE commands ends.

Next, a program example when EX2000 is used will be presented. The functions of the devices and registers used in the program are shown below.

Device or Register	Function
R0030	WRITE processing start.
R0060	Differential contact to execute the command to set BASIC-52 program No. and number of transfer words only one scan time.
R0061	WRITE command start.
R0062	End output of WRITE command.
RW004	Completion status register of WRITE command.
D00099	Register to store BASIC-52 program No. in WRITE command.
D00100	Register to store number of transfer words (500 words) in WRITE command.
D00101 to D00600	Analog data storage area.

[WRITE Command Program (EX2000)]



(1) Starts WRITE command processing when R0030 switches on.

(2) EX2000 can send 500-word data in one time by executing the WRITE command.

(3) Ends WRITE command processing when the execution of the WRITE command ends.

[WRITE Processing BASIC-52 Program]

```
10  REM *** SINGLE WRITE SAMPLE PROGRAM ***
20  BAUD 1200
30  PRINT# "*** ANALOG DATA    [V] ***"
40  CALL 18 : POP WCNT          (1)
50  FOR I=1 TO WCNT
60   CALL 6 : POP AI
70   AI=AI/200                  (2)
80   PRINT# SPC(5),AI
90  NEXT I
100 PRINT#
110 CALL 14   (3)
120 CALL 16   (4)
```

(1) Sets the number of data words written (number of data pieces) in WCNT.

(2) Displays the data on the console device connected to CH2.

(3) Notifies the EX CPU that WRITE processing has ended.

(4) Sets to the mode to queue up for command input by the EX CPU.

CAUTION Be sure to perform CALL16 in succession afterward.

6.3 WRITE with ANSWER Processing

WRITE with ANSWER processing performs the following processing.

(1) Using the WRITE sequence and WRITE command, the EX CPU writes data in the ASCII module.

(2) The ASCII module processes arithmetic operations during the BASIC-52 program designated by the WRITE sequence/WRITE command using this data.

(3) The ASCII module sets the arithmetic results obtained in (2) in the READ buffer memory as "ANSWER."

(4) The EX CPU receives the ANSWER data set in (3).

This ends one cycle of processing.

[Program Example]

The EX CPU writes the values of the calendar registers, current production counts of the production lines A, B, and C, and the target production counts of the production lines A, B, and C to the ASCII module. The ASCII module displays the calendar register values on the console device as the present date and time. It also displays the current production counts. The operator can change the preset values for the target production counts, and the changed target counts are set in the READ buffer memory as "ANSWER". The EX CPU receives the ANSWER data. When this WRITE processing with ANSWER is executed again, the change is reflected.

A program example on the EX CPU side when the ASCII module is used as an I/O module is shown below. The I/O assignment at this time is as follows:

Slot No.	I/O Type	Register No.	Assignment	
0	X+Y 04W	XW 000	ASCII module	Status register
		XW 001		READ data register
		YW 002		Command register
		YW 003		WRITE data register
1	X 01W	XW 004	DI-6261	

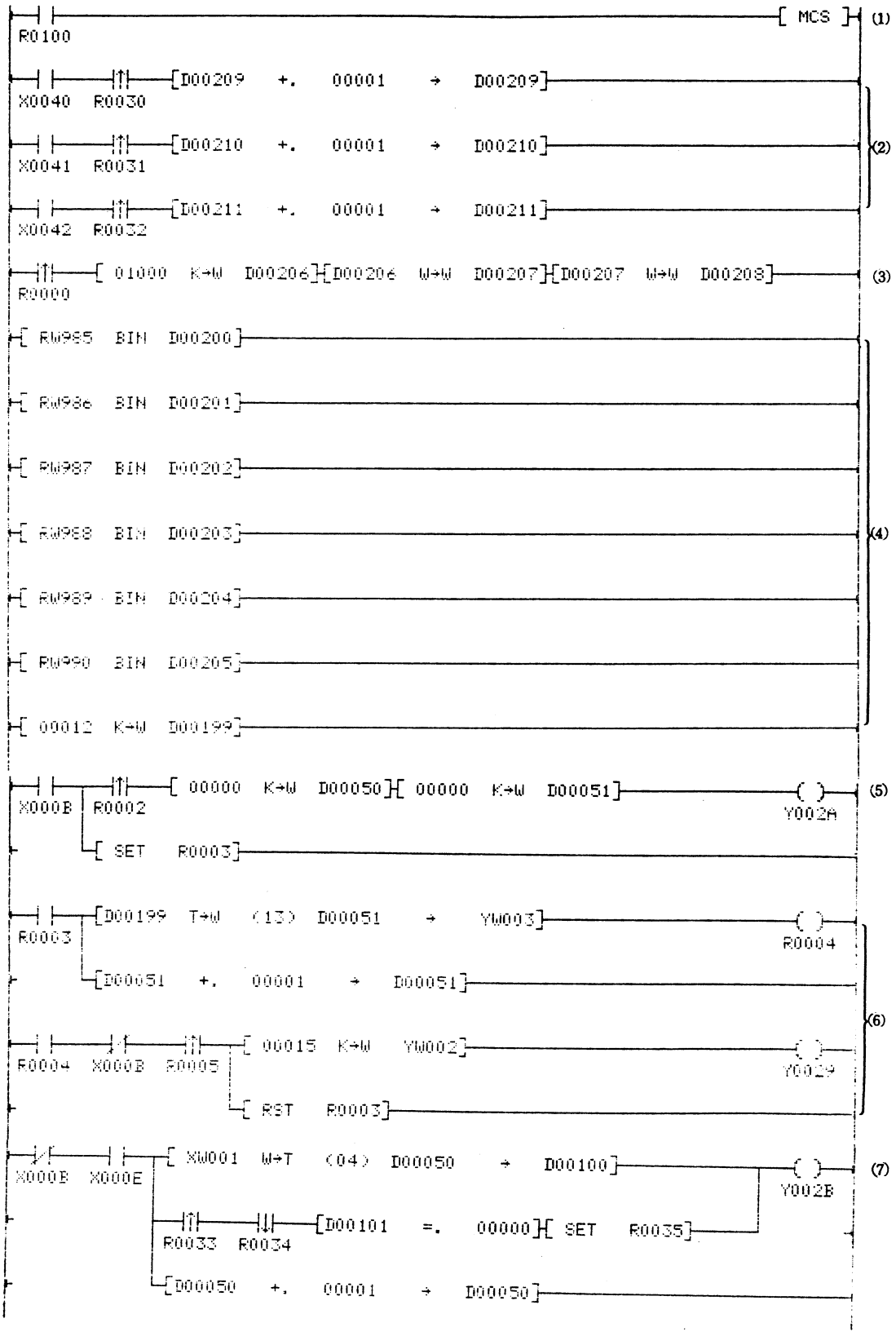
This sequence program activatges the ROM15 program on the ASCII module side.

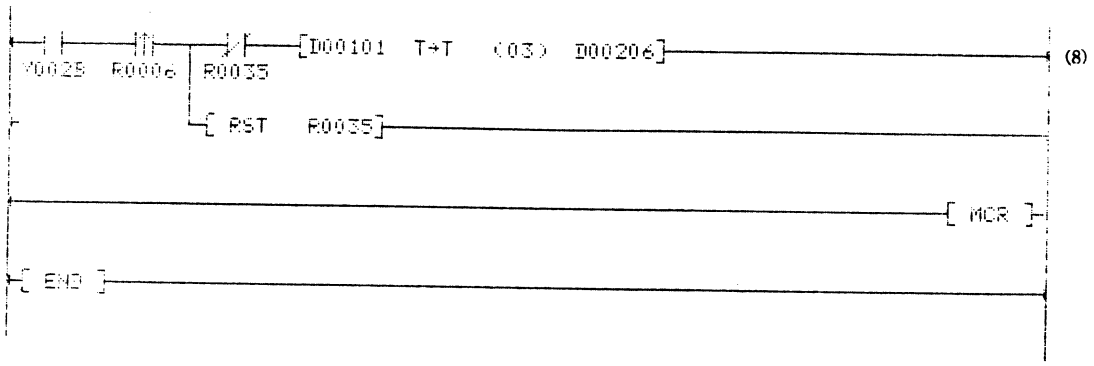
The functions of the devices and registers used in the programs are shown below.

Device or Register	Function
X0040	Data input device for counting production quantities of Production Line A.
X0041	Data input device for counting production quantities of Production Line B.
X0042	Data input device for counting production quantities of Production Line C.
R0000	Differential contact to execute the command to set goal quantity from D00206 to D00208 only one scan time.
R0002	Differential contact to execute command to clear the pointer and to switch on the WRITE start device of the command register for the ASCII module only one scan time.
R0003	Data write being executed.
R0004	Data write end.
R0005	Differential contact to execute the command to set BASIC-52 program No. and to switch on the WRITE data set end device for the ASCII module only one scan time.
R0006	Differential contact to execute goal quantity change processing only one scan time.
R0030	Differential contact to count up production quantity counter from D00209 to D00211 each time the data input device for production quantity counting is set to on.
R0031	
R0032	

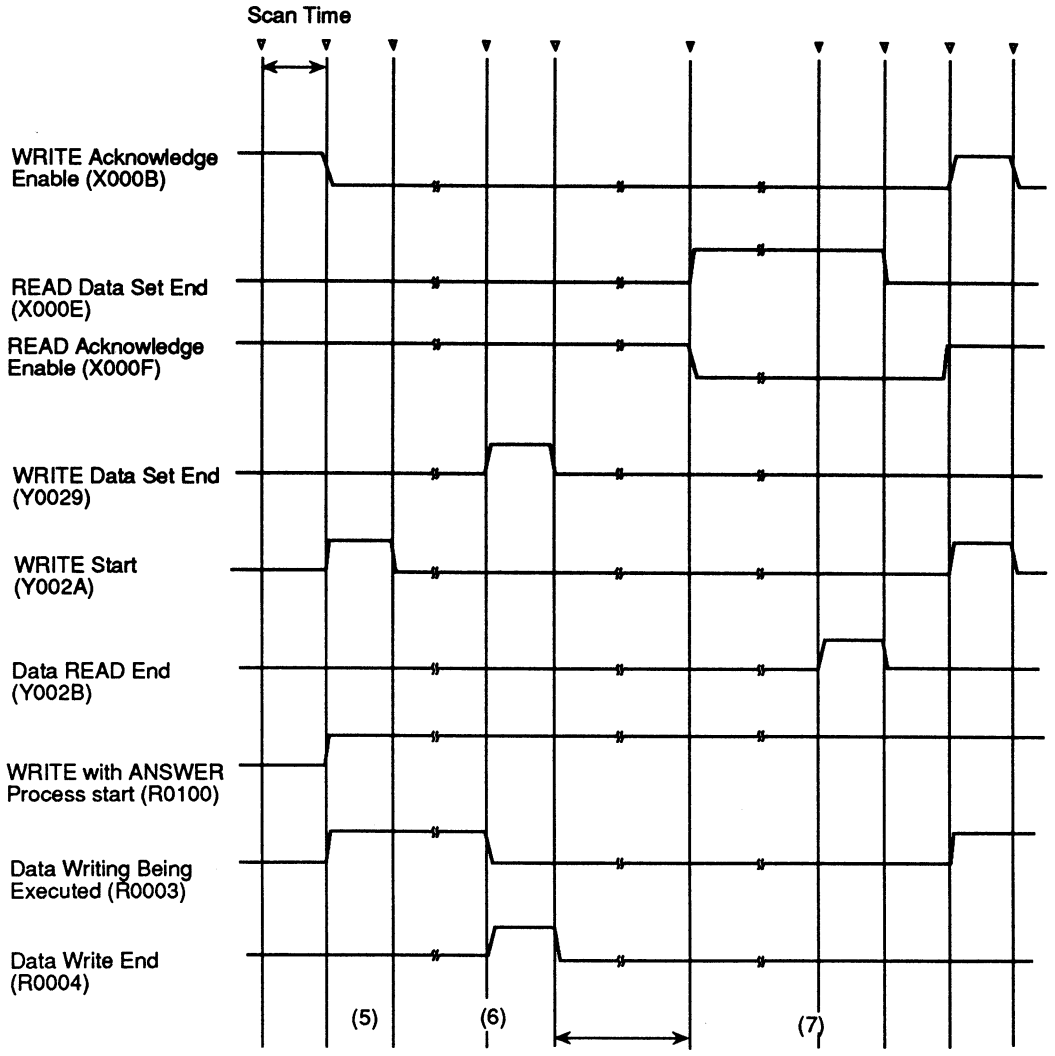
Device or Register	Function
R0033	Differential contact to execute the set command of the device (R0035) to decide whether or not to change goal quantity after reading ANSWER data only one scan time.
R0034	
R0035	Set to on if ANSWER data is "0."
R0100	Activation of WRITE with ANSWER processing.
D00050	Pointer of ANSWER data storage area.
D00051	Pointer for data to be written.
D00100	Register to store a number of ANSWER data words.
D00101 to D00103	ANSWER data storage area.
D00199	Register to store a number of data words to be written (12 words).
D00200 to D00211	Output data (WRITE data) storage area.

[WRITE Sequence Processing with ANSWER Program]





The timing chart of this program is shown below.



(1) The WRITE with ANSWER sequence processing starts when R0100 is set to on.

(2) The production quantities of production lines A, B, and C are set as WRITE data from D00209 to D00211.

(3) The target quantities of production lines A, B, and C are set as WRITE data from D00206 to D00208.

(4) Sets calendar register values as WRITE data in D00200 to D00205.

(5) The WRITE Start device is set if the WRITE Acknowledge Enable device is '0.'

(6) Then writes data to the ASCIi module, sends the No. of the BASIC-52 program to be started after writing data, and sets the WRITE data set end device.

(7) If ANSWER data from the ASCII module is set, receives the data and stores them from D00100 to D00103, then sets the data READ end device.

(8) The data received as ANSWER is reflected as the target quantities of the lines.

A program example on the EX CPU side when the ASCII module is used as an option module is shown.

The I/O assignment at this time is as follows:

Slot No.	I/O Type	Register No.	Assignment
0	OPT	——	ASCII module
1	X 01W	XW000	DI-6261

This sequence program activates the ROM15 program on the ASCII module side.

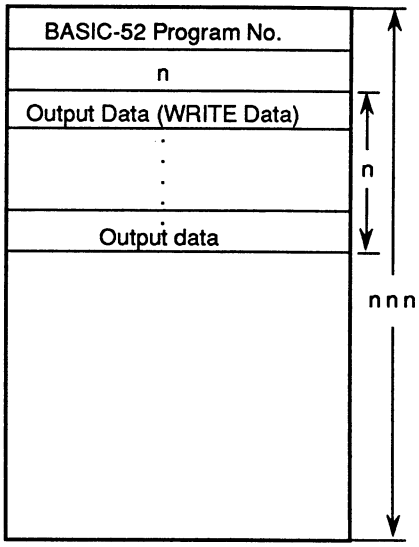
The functions of the devices and registers used in the program are shown below.

Device or Register	Function
X0000	Data input device for counting production quantities of Production Line A.
X0001	Data input device for counting production quantities of Production Line B.
X0002	Data input device for counting production quantities of Production Line C.
R0000	Differential contact to execute the command to set goal quantity in D00206 to D00208 only one scan time.
R0030	Differential contact to count up production quantity counters D00209 to D00211 each time data input device for counting production quantity is set to on.
R0031	
R0032	
R0100	WRITE with ANSWER processing start.
R0101	End output of WRITE with ANSWER command.
R0103	Device to decide whether or not to perform processing to reflect ANSWER data.
RW011	Completion status register of WRITE with ANSWER command.
D00198	Register to store BASIC-52 program No.
D00199	Register to store number of transfer words (WRITE and ANSWER data).
D00200 to D00211	Area to store output data (WRITE data).
D00212 to D00214	Area to store ANSWER data.

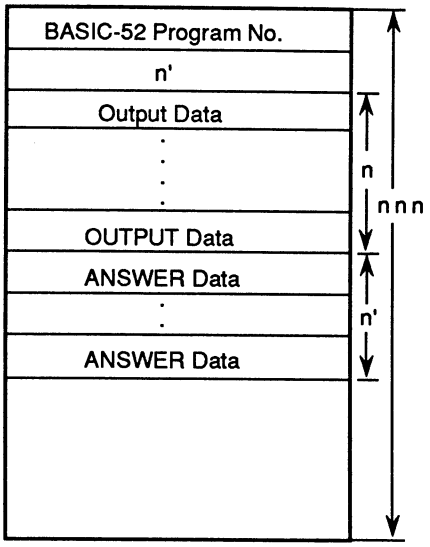
As the list of device and register functions in the preceding page indicates, ANSWER data from the ASCII module is stored in the EX CPU as if to follow the WRITE data of the parameter table in the WRITE command during processing of WRITE with ANSWER.

Using a parameter table diagram, this can be illustrated as follows:

<At Start of WRITE with ANSWER Command>



<At End of Execution of WRITE with ANSWER Command>

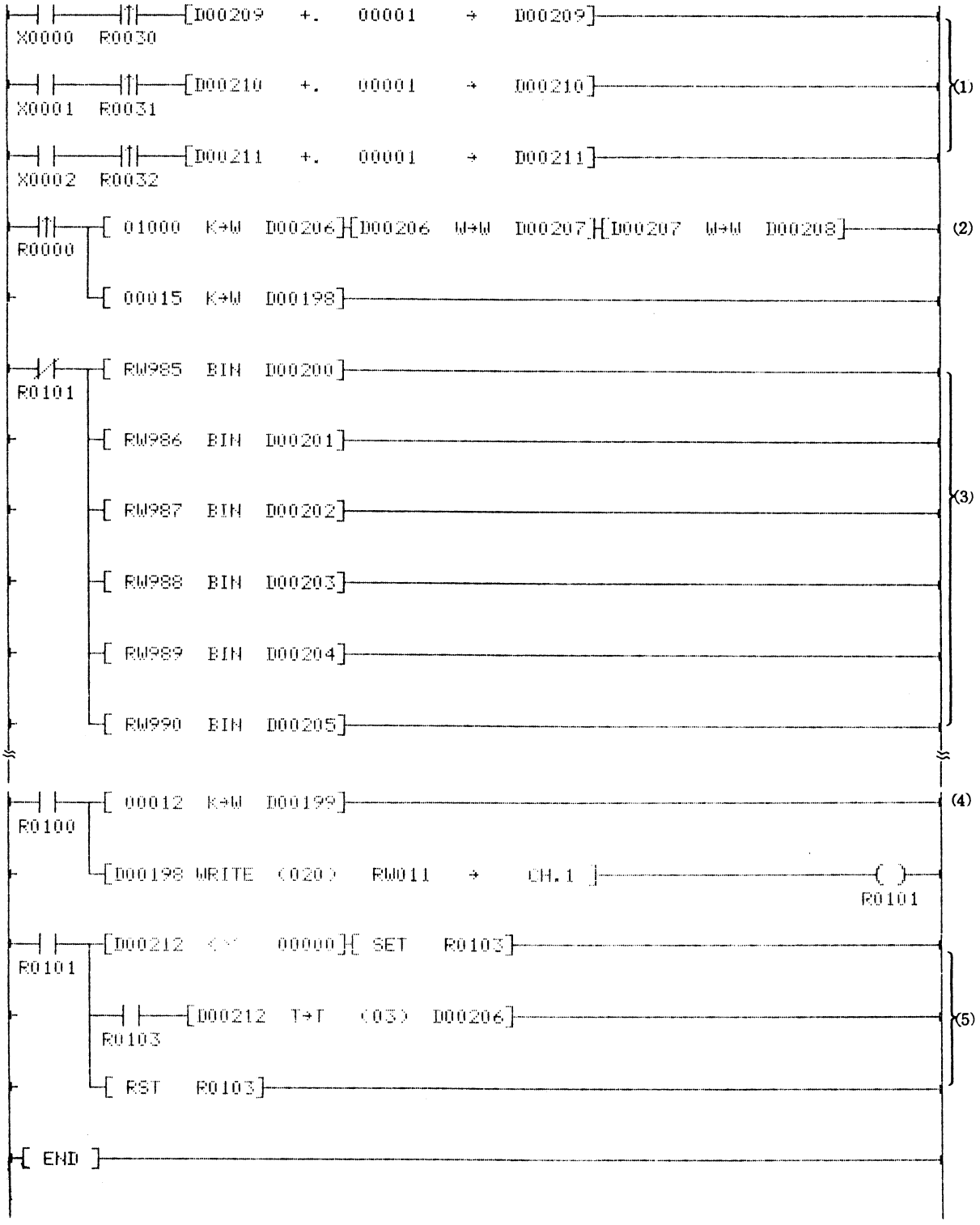


nnn: Parameter table size set during WRITE command execution.

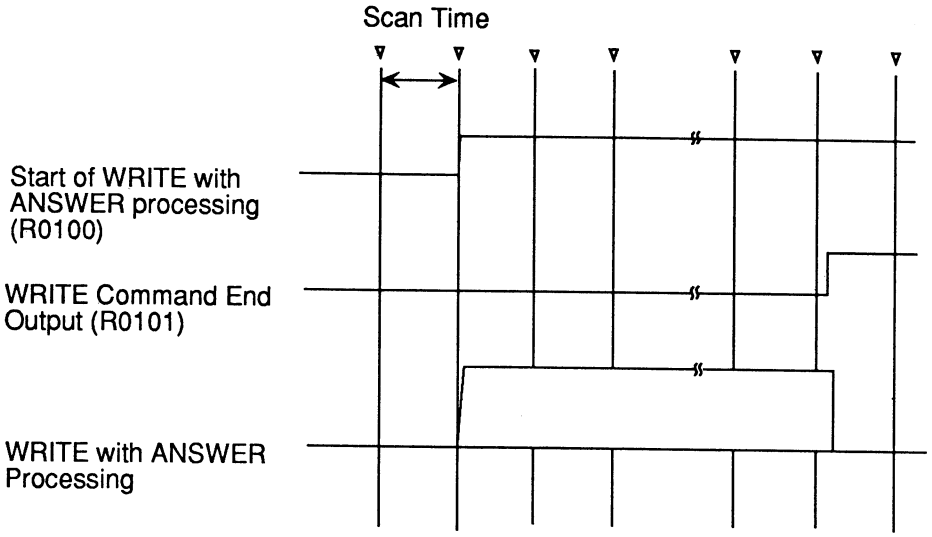
n: Number of output data words sent to ASCII module.

n': Number of ANSWER data words received from ASCII module.

[WRITE Command with ANSWER Program]



The timing chart of this program is shown below.



(1) Sets production quantities of Production Lines A, B and C in D00209 to D00211 as WRITE data.

(2) Sets goal quantities of Production Lines A, B and C in D00206 to D00208 as WRITE data.

(3) Sets calendar register values in D00200 to D00205 as WRITE data.

(4) Starts WRITE command with ANSWER processing when R0100 switches on.

(5) Reflects data received in D00212 and after as ANSWER data as the goal quantities of the lines.

[WRITE WITH ANSWER BASIC-52 Program]

```
10  REM *** ANSWER WRITE SAMPLE PROGRAM ***
20  PRINT USING(0)
30  DIM B(20)
40  PRINT "*** CONTROL PRODUCT COUNTS (PRODUCT 'A','B','C') ***"
50  FOR I=0 TO 11                                              }
60   CALL 2                                                    } (1)
70   POP B(I)                                                  }
80  NEXT I                                                     }
90  PRINT TAB(5),"DATE",B(0),"-",B(1),"-",B(2)," ",B(3),":",B(4),":",B(5)
100 PRINT
110 PRINT "    PRODUCT COUNTS (PRESENT COUNT & TARGET COUNT)    "
120 PRINT USING(#####)
130 PRINT TAB(5),"A=",B(9),"(",B(6),")"
140 PRINT TAB(5),"B=",B(10),"(",B(7),")"
150 PRINT TAB(5),"C=",B(11),"(",B(8),")"
160 PRINT
170 PRINT " CHANGE TARGET? (1:YES 2:NO)"
180  D=GET
190  IF D=31H GOTO 210
200  IF D=32H GOTO 340 ELSE GOTO 180
210 REM *** ROUTINE FOR CHANGING TARGET VALUES ***
220 INPUT "A= ",B(12)
230 INPUT "B= ",B(13)
240 INPUT "C= ",B(14)
250 CALL 19                (2)
260 FOR I=12 TO 14                                             }
270  PUSH B(I)                                                 } (3)
280  CALL 10                                                   }
290 NEXT I                                                     }
300 PRINT "CHANGE COMPLETE"
310 CALL 20                          (4)
320 CALL 15                               (5)
330 CALL 16                                    (6)
340 REM *** ROUTINE FOR NO CHANGING ***
350 PRINT "NO CHANGE"
360 B(12)=0
370 CALL 19
380 PUSH B(12)
390 CALL 10
400 GOTO 310
```

(1) Sets calendar data and production line information written by the EX CPU in B(0) to B(11).

(2) Declares starting of ANSWER data set.

(3) Sets ANSWER data in the READ buffer memory.

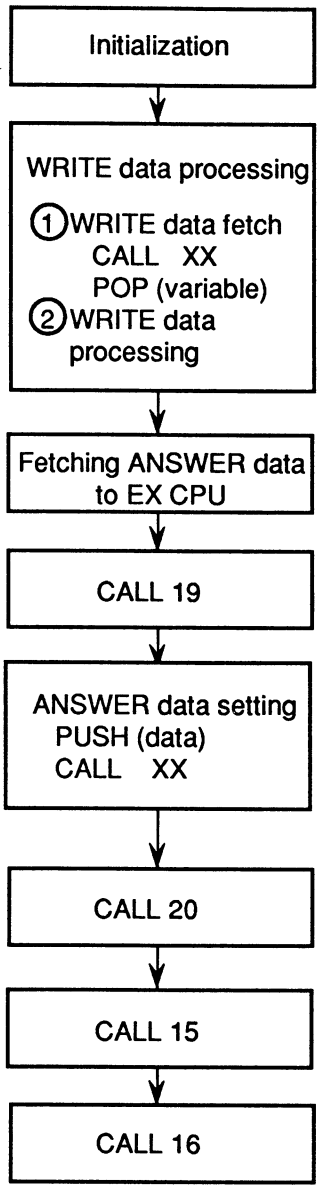
(4) Notifies the EX CPU that setting of ANSWER data has been completed.

(5) Notifies the the EX CPU that processing of WRITE with ANSWER has been finished.

(6) Sets to the mode to queue up for a command to be input by the EX CPU.

CAUTION Be sure to perform CALL16 in succession afterword.

The operating method and timing of CALL subroutines during the execution of a BASIC-52 program for WRITE with ANSWER processing are as follows:



6.4 Continuous READ Processing

The continuous READ processing performs the following processing.

(1) Using the READ sequence and READ command, the EX CPU requests the ASCII module to start a specified BASIC-52 program.

(2) The ASCII module fetches data from the keyboard of the console device, bar code reader or other device, during a BASIC-52 program activated as mentioned in (1). The ASCII module sets the data in the READ buffer memory.

(3) The EX CPU reads the set data. The EX CPU processes steps (2) and (3) continuously afterward during the READ sequence until the READ acknowledge enable device is set to '1.'
During the READ command execution, the EX CPU continues processing steps (2) and (3) by changing setting of the DATA READY device of the completion status register from '1' to '0.'

[Program Example]

The console device connected to CH2 of the ASCII module displays a message when the EX CPU starts a BASIC-52 program. This message tells the number of continuous READ cycles. Data sent to the EX CPU is 250 words for one cycle of processing, and this cycle is repeated for the number of opration cycle.
During reading, bar code data of a product consisting of 13 numerals is read as a character string by the bar code reader connected to CH1 of the ASCII module.

The lower three digits of character data in this character string are converted into numerical data and are set in the READ buffer memory as READ data. The EX CPU reads the data and stores them in the data registers.

A program example on the EX CPU side when the ASCII module is used as an I/O module will be shown. The I/O assignment at this time is as follows:

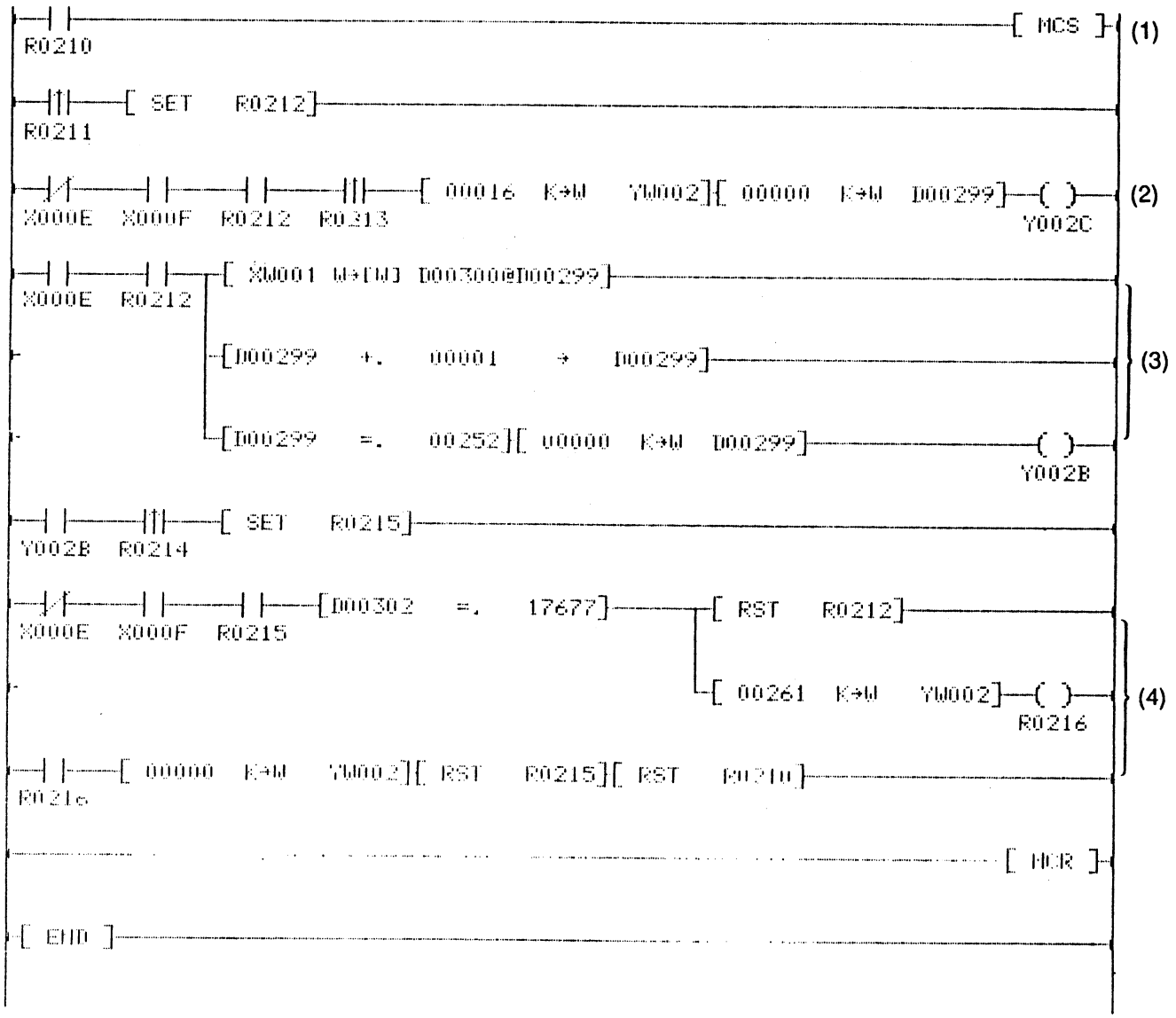
Slot No.	I/O Type	Register No.	Assignment	
0	X+Y 04W	XW 000	ASCII module	Status register
		XW 001		READ data register
		YW 002		Command register
		YW 003		WRITE data register

This sequence program activates the ROM16 program on the ASCII module.

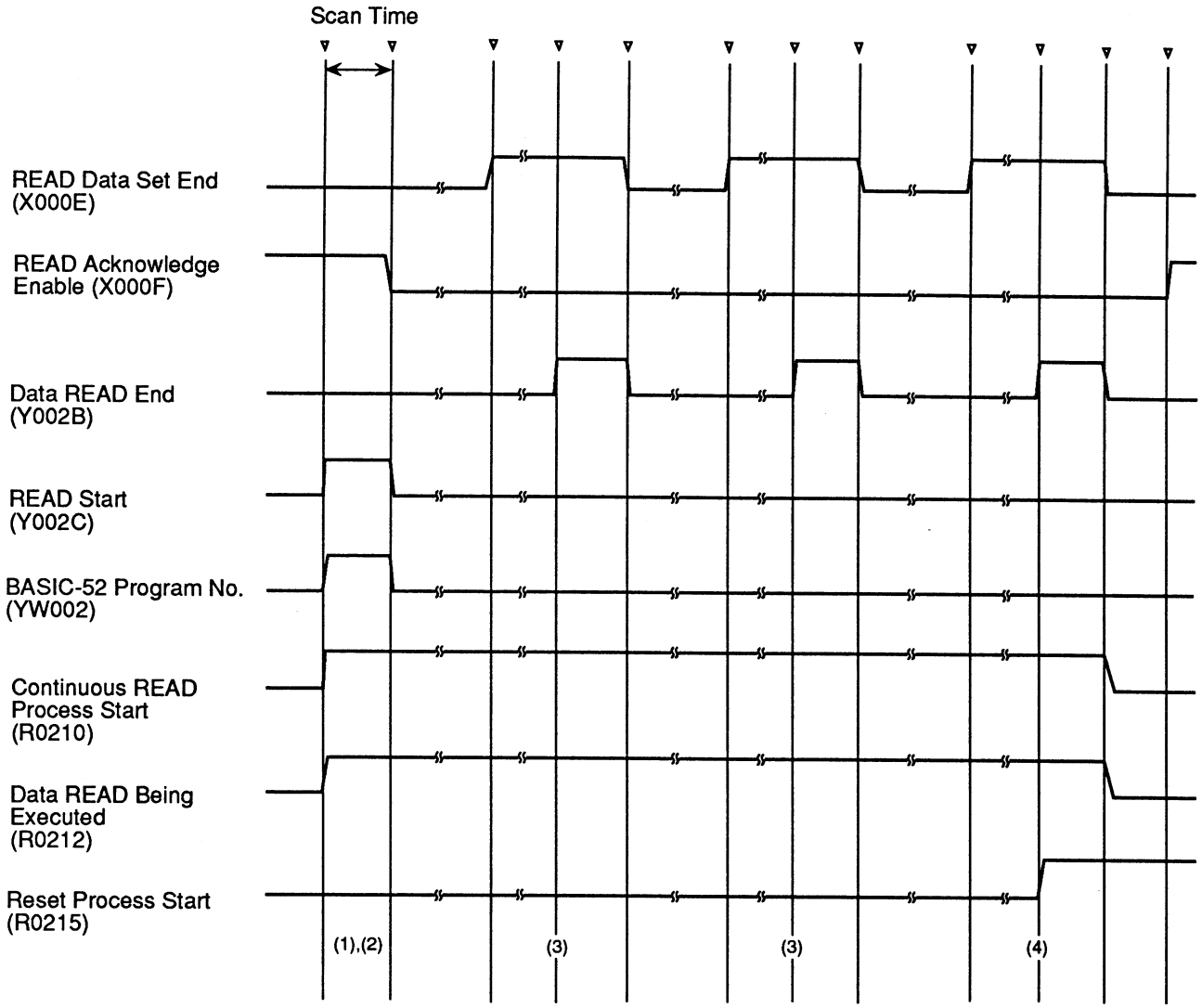
The functions of the devices and registers used in the program are as follows:

Device or Register	Function
R0210	Continuous READ processing start.
R0211	Differential contact to execute the command to set the Data READ Being Executed Device only by one scan.
R0212	Data READ being executed.
R0213	Differential contact to execute the command to transfer BASIC-52 program No., to clear the READ data storage area pointer, and to set the READ start device of the ASCII module command register only one scan time.
R0214	Differential contact to execute the command to set the reset process start device only one scan time.
R0215	Reset process start.
R0216	Reset process end.
D00299	Pointer of input data (READ data) storage area.
D00301	Register to store a number of words (250 words) of input data (READ data).
D00302 to D00551	Input data (READ data) storage area.

[Continuous READ Sequence Program]



The timing chart of this program is shown in the next page.



(1) Starts the continuous READ sequence when R0210 switches on.

(2) If the READ data set end device is '0' and READ acknowledge enable device is '1', sends the No. of the BASIC-52 program to be started to the ASCII module, then sets the READ START device.

(3) If the READ data set end device becomes '1', reads READ data set in the ASCII module from D00301 to D00551 and sets the data READ end device.

(4) If the read data shows "end of data transfer" from the ASCII module, the ASCII module is reset and continuous READ sequence is finished.

A program example of the EX CPU side when the ASCII module is used as an option module is shown below.

The I/O assignment at this time is as shown below.

Slot No.	I/O Type	Register No.	Assignment
0	OPT	———	ASCII module

This sequence program activates the ROM16 program on the ASCII module side.

The functions of the devices and registers used in the program are shown below.

Device or Register	Function
R0210	Continuous READ processing start.
R0211	Differential contact to exeute the command to set BASIC-52 program No. only one scan time.
R0212	Continuous READ command start.
R0213	End output of continuous READ command.
R0214	Reset READ command start.
R0215	End output of reset READ command.
RW022	Completion status register of continuous READ command.
RW023	Completion status register of reset READ command.
R022E	Data Ready device of completion status register of continuous READ command.
D00300	BASIC-52 program No. storage register.
D00301	Register to store number of words (250 words) of input data (READ data).
D00302 to D00551	Intput data (READ data) storage area.

The Data Ready Device of the Continuous READ Command Completion Status Register is used to perform continuous READ processing using the READ command.

(1) This device will be set to "1" if the EX CPU reads all READ data set by the ASCII module.

(2) The EX CPU fetches data from the input data storage area during the execution of a ladder sequence program and resets this device to "0." The ASCII module does not perform READ data setting after the second time until this device becomes "0" during the execution of a BASIC-52 program.

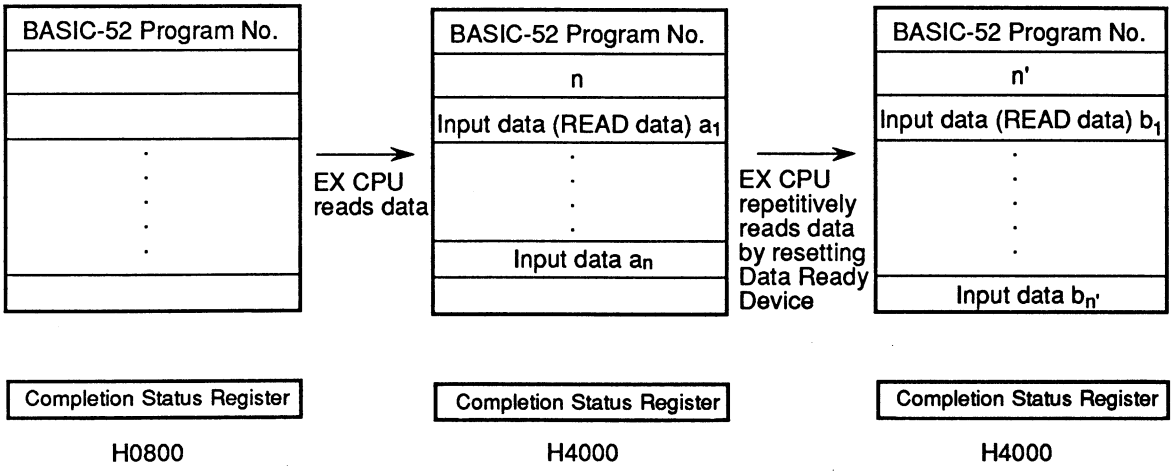
(3) The ASCII module restarts setting READ data if this device becomes "0."

(4) After finishing setting all data, the ASCII module notifies it to the EX CPU by the CALL 20 statement of the BASIC-52 program.

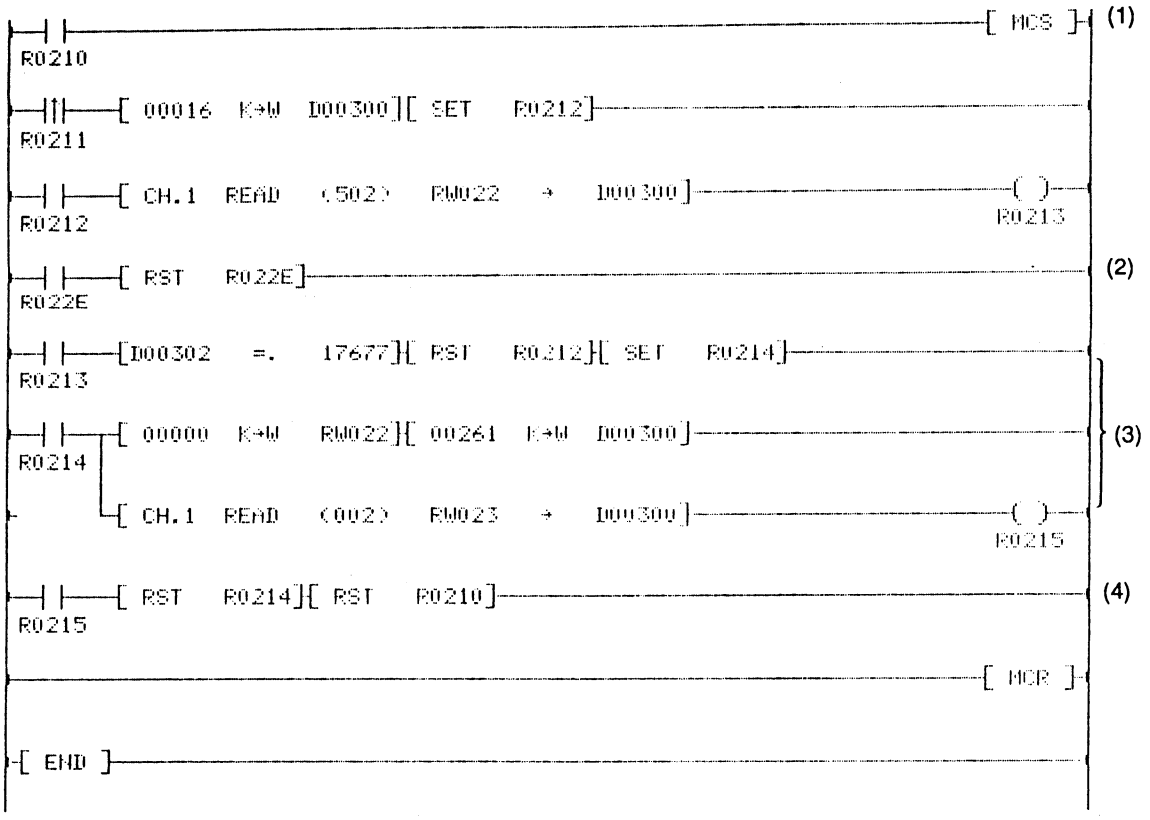
(5) This device is reset to "1" when the EX CPU reads all the data.

Continuous READ processing is performed by repeating Steps (2) to (5). The data table values in the READ command for continuous READ vary as follows:

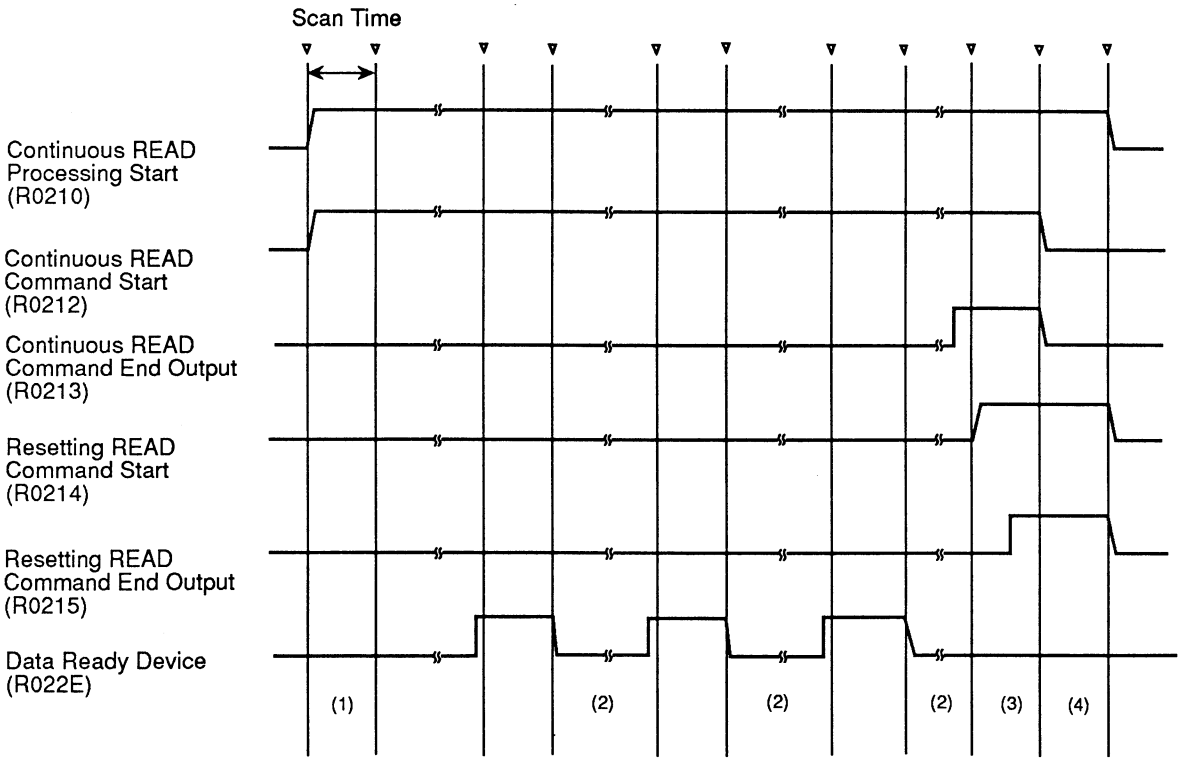
<During Activition of Continuous READ Command>



[Continuous READ Command Program]



The timing chart of this program is shown below.



(1) Starts the continuous READ commands when R0210 switches on.

(2) Data can be read continuously by changing setting of the DATA READY device of the completion status register from '1' to '0.'

(3) Resets the ASCII module if the read data shows "end of data transfer" from the ASCII module.

(4) The continuous READ sequence is finished.

[Continuous READ BASIC-52 Program]

```
10  REM *** CONTINUOUS READ SAMPLE PROGRAM ***
20  BAUD 4800
30  STRING 20,15
40  CNT=10
50  PRINT# "Data counts to EX =250*",CNT
60  PRINT# "***  DATA FROM BAR CODE READER  ***"
70  C=0
80  DO
90   C=C+1
100  CALL 19
110  FOR I=1 TO 250
120   $(0)=""
130   FOR J=1 TO 15
140    G=GET
150    IF G=0 THEN GOTO 140
160    ASC($(0),J)=G
170   NEXT J
180   PRINT# "DATA CODE =",$(0),"    ",
190   REM $(0) --> VARIABLE
200   A=0 : B=0
210   FOR J= 15 TO 13 STEP -1
220    A=ASC($(0),J)-30H
230    B=B+A*10**(15-J)
240   NEXT J
250   PRINT# B,CR
260   PUSH B
270   CALL 10
280  NEXT I
290  CALL 20
300  ANS=XBY(7400H) .AND.01H
310  IF ANS=1H GOTO 300
320 UNTIL C=CNT
330 PRINT# "Completed Data sending to EX."
340 CALL 19
350 $(0)="E"
360 CALL 8
370 CALL 20
380 CALL 13
390 CALL 16
```

(1) (2) (3) (4) (5) (6) (7)

(1) Reads and displays bar code data on the console device connected to CH2.

(2) Sets numerical data in the READ buffer memory.

(3) Notifies the EX CPU that setting of 250 pieces of data has been finished.

(4) Waits by looping until the EX CPU finishes reading data.

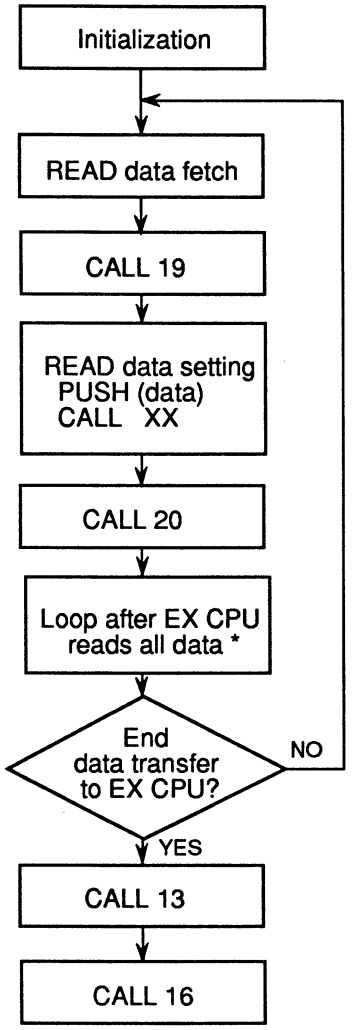
(5) Sends data indicating end of data transfer to the EX CPU.

(6) Notifies the EX CPU that continuous READ processing has been finished.

(7) Sets to the mode queueing up for input.

CAUTION Be sure to perform CALL16 in succession afterword.

The operating method and timing of the CALL subroutines in the BASIC-52 programs for continuous READ processing are as follows:



* Details of this processing are described below.

Use the flags stored at Address 7400H of the external memory of the ASCII module to check if the EX CPU has read all data during the execution of a BASIC-52 program.

Address 7400H	7	6	5	4	3	2	1	0	(Bits)

↑

Flag of this bit is used.

Bit 0 at Address 7400H has the following function:

Bit	Function
0	"1" is set if the EX CPU can read data. This status is maintained while the EX CPU reads data. "0" will be set when the EX CPU ends reading the data. Regard this bit as reflecting the status of the READ data set end device of the status register if the ASCII module is used as an I/O module.

Whether or not the EX CPU has finished reading data can be known by checking if Bit 0 at Address 7400H has become "0" during the execution of a BASIC-52 program. The program example shown in page 109 makes this decision by processing Step (4) (lines 300 and 310).

6.5 READ/WRITE Simultaneous Execution Processing

The READ/WRITE simultaneous execution processing performs the following processing.

If READ processing is initiated first:

(1) Using the READ sequence and READ command, the EX CPU sends requests to the ASCII module to start specified BASIC-52 program.

(2) The ASCII module fetches data from the keyboard of the console device, bar code reader or other device, during a BASIC-52 program started as mentioned in (1).

(3) The READ data set by the ASCII module is input non-periodically. During this time the EX CPU "queues" and processes writing utilizing this period of time. The EX CPU writes data in the ASCII module using the WRITE sequence and WRITE command. Then designates a BASIC-52 program No. that processes writing. The program No. designated at this time must be identical to that designated in READ processing.

(4) The ASCII module processes the data written by the EX CPU during a BASIC-52 program.

(5) The EX CPU reads the set data when the ASCII module ends data fetching and data setting in the buffer memory during data reading mentioned in (2).

This ends one cycle of READ/WRITE processing.

If WRITE processing is initiated first:

(1) Using the WRITE sequence and WRITE command, the EX CPU writes data in the ASCII module and sends requests to start specified BASIC-52 program.

(2) The ASCII module processes this data during a BASIC-52 program started as mentioned in (1).

(3) Using the READ sequence or READ command, the EX CPU makes the ASCII module start the process of reading while the ASCII module processes written data. Then the EX CPU can read data inputted by peripheral device connected to the ASCII module. The BASIC-52 program No. for data read processing must be identical to that designated in WRITE processing.

(4) The ASCII module fetches data from the keyboard of the console device, bar code reader, or other device, and sets the data in the buffer memory. During this time, the written data is processed in parallel.

(5) The EX CPU reads the set data. When the ASCII module ends processing of written data, one READ/WRITE processing ends.

[Program Example]

The EX CPU first starts a BASIC-52 program using the READ sequence/READ command. During the BASIC-52 program execution, data is input through the bar code reader connected to CH1 of the ASCII module. Based on this data, the ASCII module sets the data to be read by EX CPU in the READ buffer memory. Writing is performed during this time.

The EX CPU writes the values of the calendar registers in the ASCII module, which displays the values of the calendar registers in the console device connected to CH2 as the present date and time. The ASCII module then reads bar code data of a product by the bar code reader connected to CH1, deciding whether or not the bar code data is correct and counting the numbers of products with the correct code and of products with error code data. The count data is set in the READ buffer memory for reading by the EX CPU .

A program example on the EX CPU side when the ASCII module is used as an I/O module is shown below. The I/O assignment at this time is as follows:

Slot No.	I/O Type	Register No.	Assignment	
0	X+Y 04W	XW 000	ASCII module	Status register
		XW 001		READ data register
		YW 002		Command register
		YW 003		WRITE data register

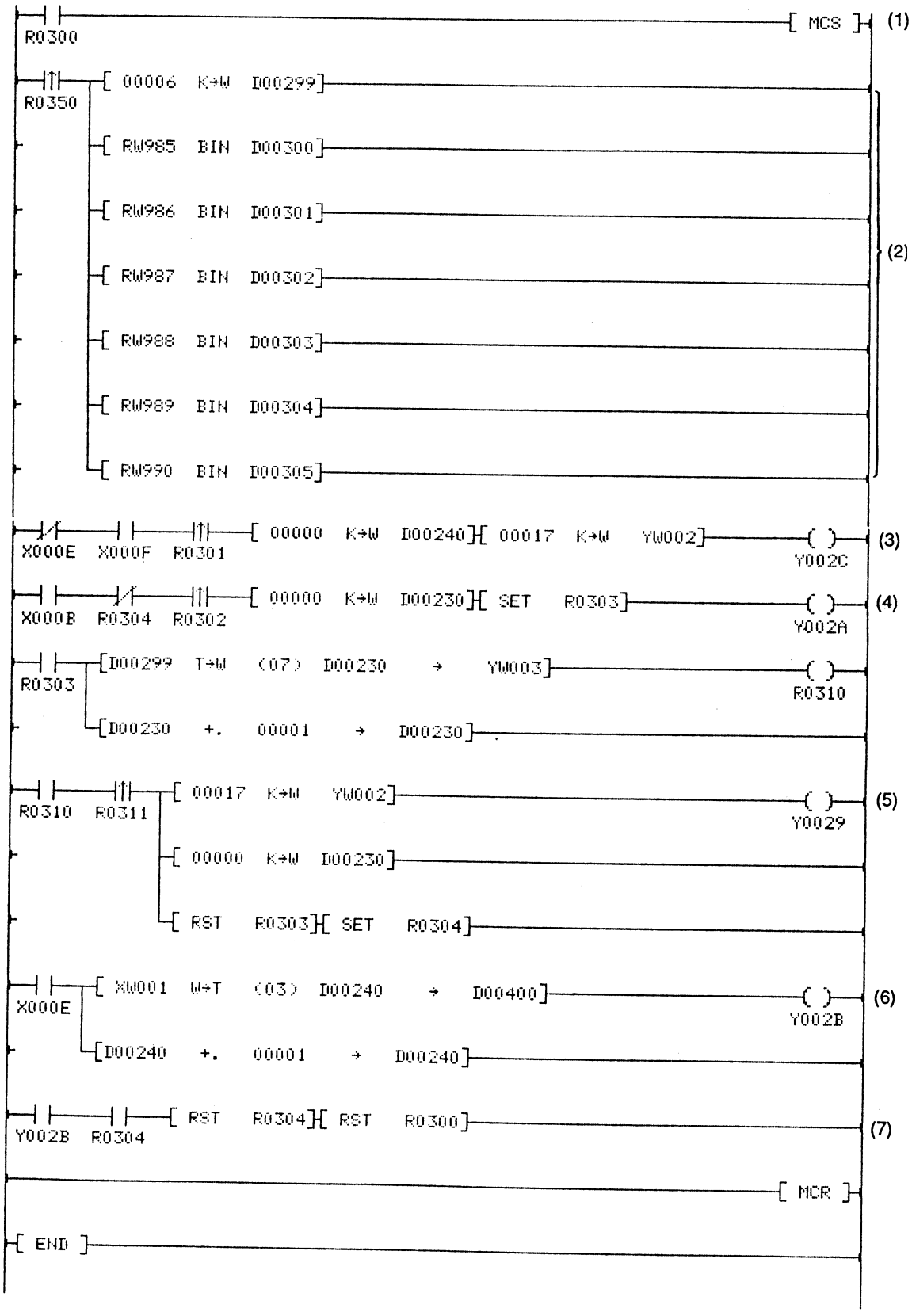
This sequence program activates the ROM17 program on the ASCII module side.

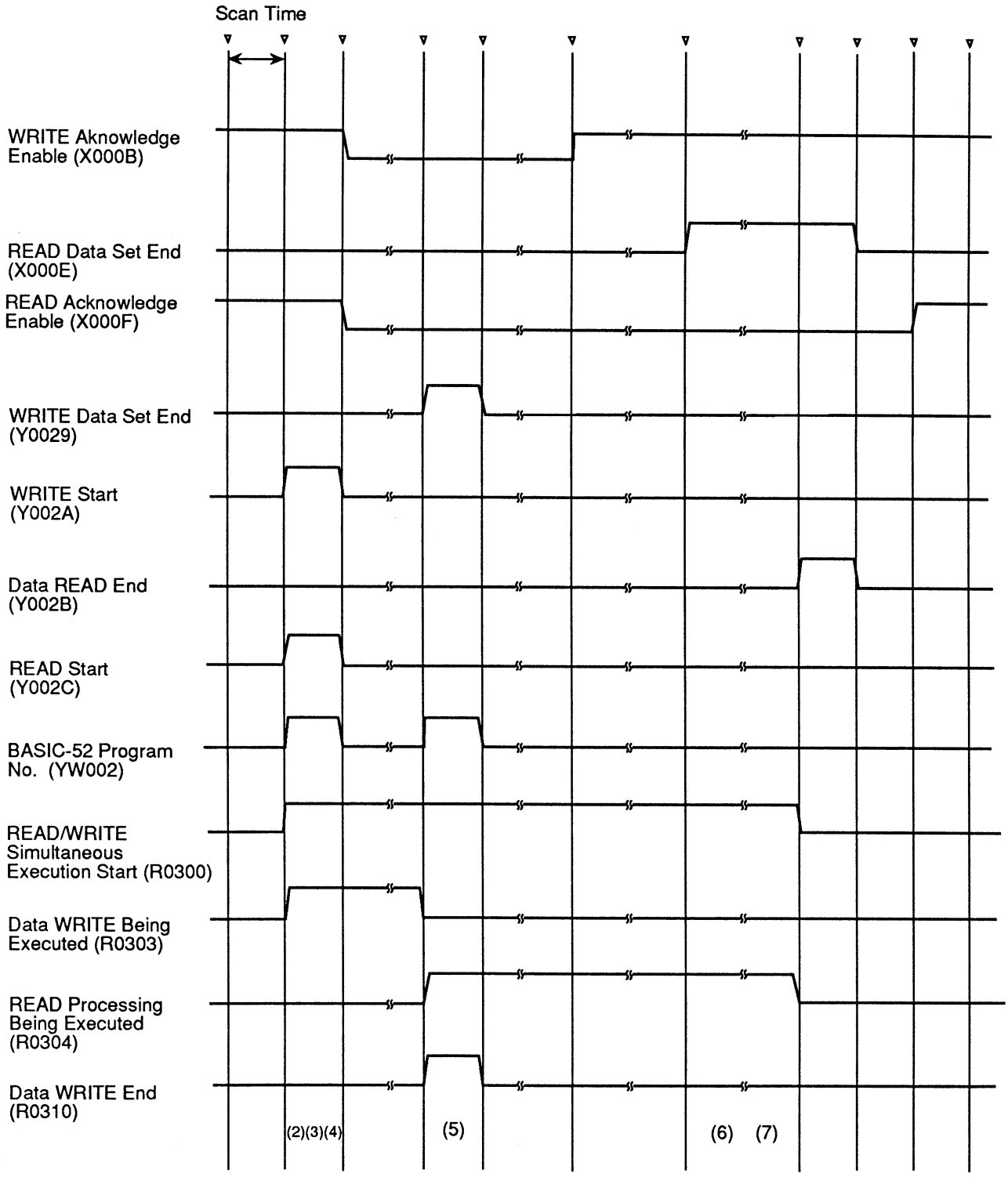
The functions of the devices and registers used in this program are shown below.

Device or Register	Function
R0300	READ/WRITE simultaneous execution activation.
R0301	Differential contact to exeute the command to clear the input data pointer, to transfer BASIC-52 program No., and to set ASCII module command register READ start device only one scan time.
R0302	Differential contact to execute the command to clear the output data pointer and to set ASCII module command register WRITE start device only one scan time.
R0303	Data write being executed.
R0304	READ processing being executed.
R0310	Data WRITE end.
R0311	Differential contact to execute the command to transfer BASIC-52 program No. and to set the WRITE data set end device only one scan time.
R0350	Differential contact to execute output data setting only one scan time.
D00230	Output data (WRITE data) pointer.

Device or Register	Function
D00240	Input data (READ data) storage area pointer.
D00299	Register to store number of words (6 words) of WRITE data.
D00300 to D00305	Output data (WRITE data) storage area.
D00400 to D00402	Input data (READ data) storage area.

[READ/WRITE Simultaneous Execution Sequence Program]




Scan Time
WRITE Aknowledge Enable (X000B)
READ Data Set End (X000E)
READ Acknowledge Enable (X000F)
WRITE Data Set End (Y0029)
WRITE Start (Y002A)
Data READ End (Y002B)
READ Start (Y002C)
BASIC-52 Program No. (YW002)
READ/WRITE Simultaneous Execution Start (R0300)
Data WRITE Being Executed (R0303)
READ Processing Being Executed (R0304)
Data WRITE End (R0310)
(2)(3)(4)
(5)
(6)
(7)

(1) Starts the READ/WRITE simultaneous execution sequence processing when R0300 switches on.

(2) Sets calendar register values as WRITE data.

(3) If the READ acknowledge enable device is '1' and READ data set end device is '0', sends the No. of the BASIC-52 program to be started to the ASCII module, then sets the READ START device.

(4) If the WRITE acknowledge enable device is '1', sets the WRITE START device and writes the data to the ASCII module.

(5) Sends the same BASIC-52 program No. as that designated already during the READ sequence after the data is finished writing, and sets the WRITE data set end device.

(6) If the READ data set end device is set, reads data and sets the data READ end device after reading data.

(7) Ends the READ/WRITE simultaneous execution sequence as both READ and WRITE sequence processing has been finished.

A program example on the EX CPU side when the ASCII module is used as an option module is shown below.

The I/O assignment at this time is as follows.

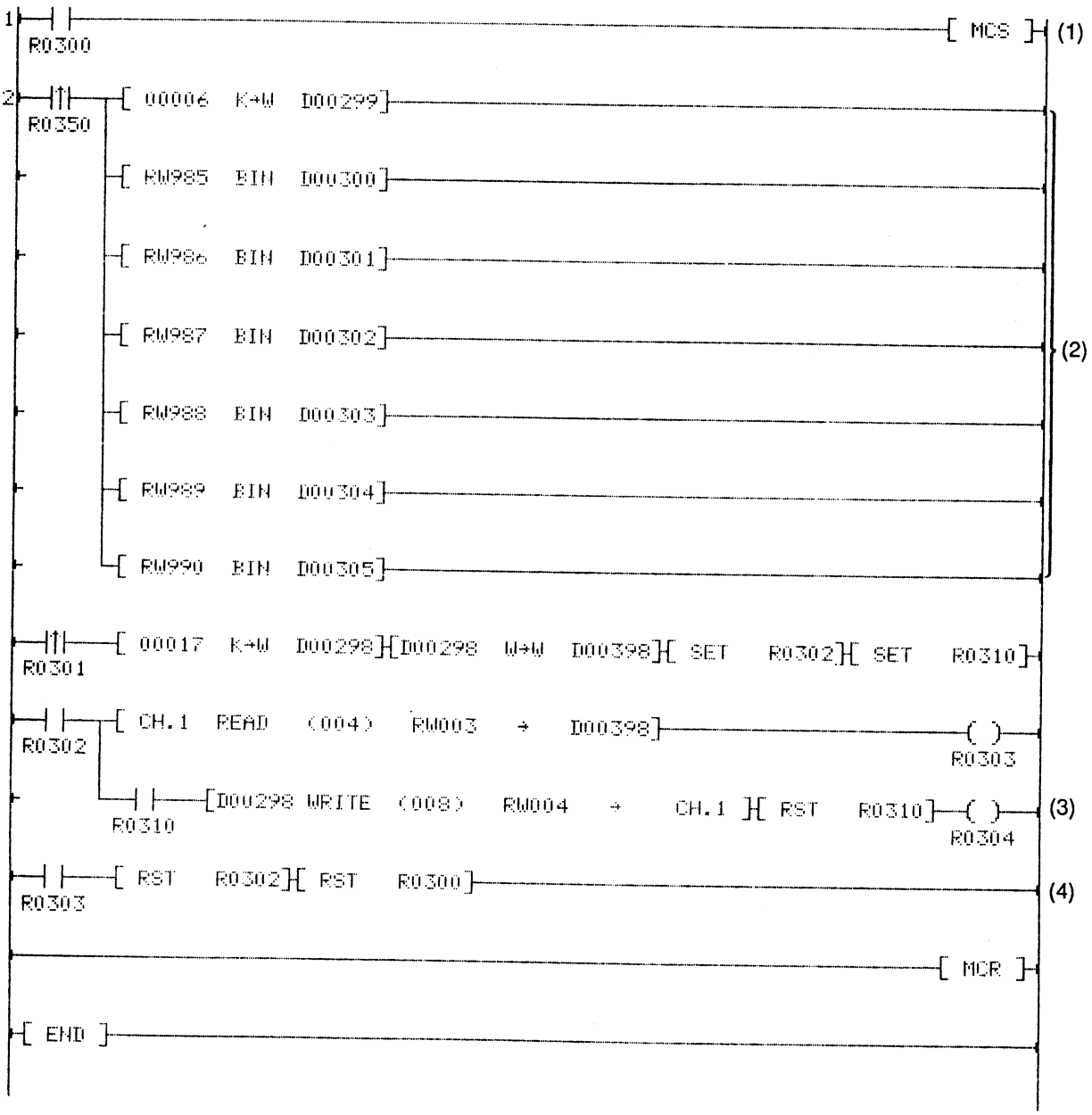
Slot No.	I/O Type	Register No.	Assignment
0	OPT	———	ASCII module

This sequence program will activates the ROM17 program on the ASCII module side.

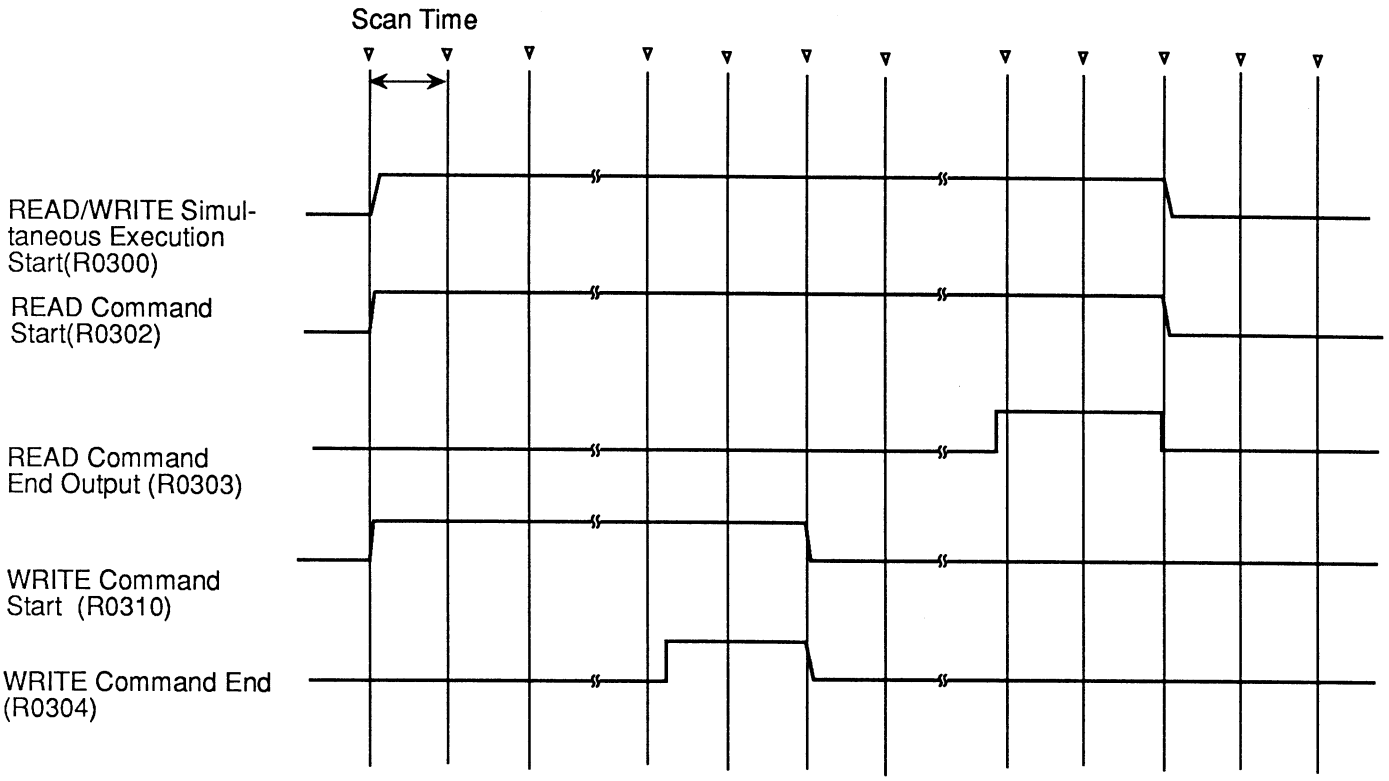
The functions of the devices and registers used in the program are shown below.

Device or Register	Function
R0300	READ/WRITE simultaneous execution activation.
R0301	Differential contact to execute the command to set BASIC-52 program No. and READ and WRITE command start devices only one scan time.
R0302	READ command start.
R0303	READ command end output.
R0304	WRITE command end and WRITE command start device reset end
R0310	WRITE command start
R0350	Differential contact to execute output data setting only one scan time.
RW003	READ command completion status register
RW004	WRITE command completion status register
D00298	Register to store BASIC-52 program No. for the WRITE command.
D00299	Register to store number of output data words.
D00300 to D00305	Output data (WRITE data) storage area.
D00398	Register to store BASIC-52 program No. for the READ command.
D00399	Register to store number of input data (READ data) words.
D00400 to D00401	Intput data (READ data) storage area.

[READ/WRITE Simultaneous Execution Command Program]



The timing chart of this program is shown below.



(1) Starts the READ/WRITE simultaneous execution command when R0300 switches on.

(2) Sets calendar register values as WRITE data.

(3) The WRITE command execution finishes first.

(4) The READ/WRITE simultaneous execution command execution is finished when READ command processing ends.

[READ/WRITE Simultaneous Execution BASIC-52 Program]

```
10  REM *** READ/WRITE SAME TIME SAMPLE PROGRAM ***
20  BAUD 4800
30  STRING 50,20
40  DIM A(10)
50  STATUS=XBY(7400H)   .AND.20H   }(1)
60  IF STATUS=0 THEN 50
70  FOR I=0 TO 5          }(2)
80   CALL 2
90   POP A(I)
100 NEXT I
110 PRINT# TAB(5),"DATE",A(0),"-",A(1),"-",A(2)," ",A(3),":",A(4),":",A(5)
120 CALL 14          (3)
130 PRINT#
140 PRINT# "***  PRODUCTION CONTROL BY BAR CODE READER  ***"
150 PRINT# " PRODUCT CODE =", : INPUT $(1) : PRINT# $(1)
160 REM DATA READ FROM BAR CODE READER
170 ERR=0 : PCNT=0
180 DO                                                          }(4)
190  ERCHK=0
200  PRINT# " INPUT CODE =",
210  INPUT $(0) : PRINT# $(0),CR,
220  I=0
230  DO
240   I=I+1
250   IF ASC($(1),I)<>ASC($(0),I) THEN ERCHK=ERCHK+1
260  UNTIL ASC($(1),I)=0DH
270  IF ERCHK=0 THEN PCNT=PCNT+1 ELSE ERR=ERR+1
280 WHILE ERR<10
290 PRINT# : PRINT# "END OF DATA GETTING..." : PRINT#
300 CALL 19          }(5)
310 PUSH PCNT
320 CALL 10
330 PUSH ERR
340 CALL 10
350 CALL 20
360 CALL 13          (6)
370 CALL 16          (7)
```

(1) Queues up by looping until the EX CPU finishes writing data.

(2) Stores the data written by the EX CPU in A(0) to A(5).

(3) Notifies the EX CPU that writing has been finished.

(4) Reads and displays bar code data on the console device connected to CH2.

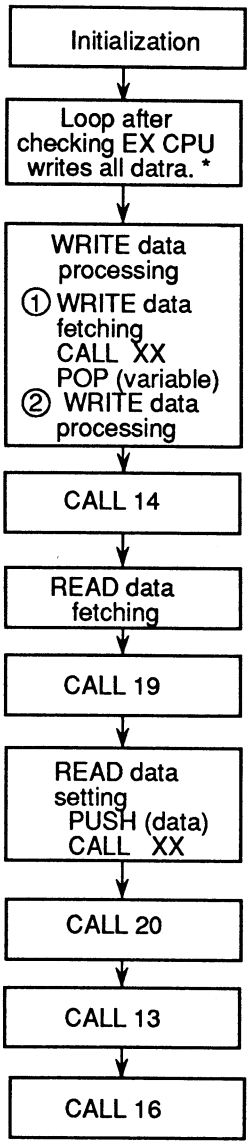
(5) Sends quantities of products of correct and error codes to the EX CPU.

(6) Notifies the EX CPU that READ/WRITE simultaneous execution processing has been finished.

CAUTION Be sure to perform CALL16 in succession afterword.

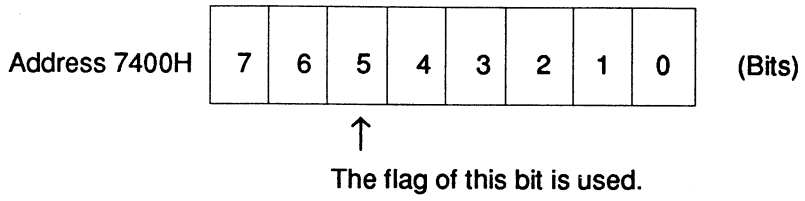
(7) Sets to the mode queuing up for input of a command by the EX CPU.

The operating method and timing of the CALL subroutines during the execution of the READ/WRITE simultaneous execution processing are described. The program example processes data written earlier after activating READ processing. This is illustrated as follows.



* Details of this processing are explained below.

The flags stored at Address 7400H of the external memory of the ASCII module are used to check whether or not the EX CPU has written all the data during the execution of the BASIC-52 programs.



Bit 5 at Address 7400H has the following function:

Bit	Function
5	Set to "1" when the EX CPU writes all data with the ASCII module. Regard as bit that reflect the status of WRITE data set end device of the command register if the ASCII module is used as an I/O module.

Whether or not the EX CPU has finished writing data can be determined by checking whether or not Bit 5 of Address 7400H has become "1" during the execution of a BASIC-52 program. The program example shown in page 122 makes a decision based on processing of (1) (50 and 60th lines).

6.6 ROM Program Automatic Execution Mode

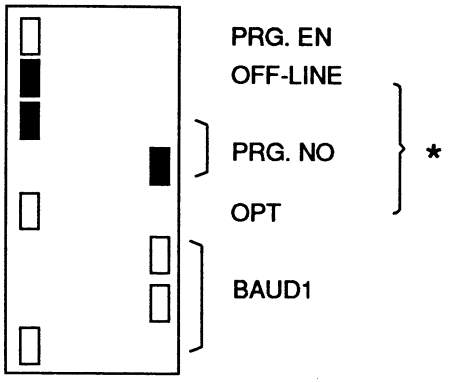
The mode to automatically execute a specified BASIC-52 program after turning on power to the ASCII module can be set up by setting the OFF-LINE and PRG. NO. setting switches in pair on the DIP switch on the front panel of the ASCII module.

[Program Example]

Store the following program in ROM1 of the ASCII module:

```
10    REM *** PROGRAM COUNTS CHECK ***
20    PRINT "**** PROGRAM COUNTS CHECK ****"
30    PRINT "PROGRAM COUNTS=", XBY (7408H)
40    PRINT "PROGRAM EDIT (1) or EX INTERFACE (2)  ?"
50    ANS=GET
60    IF ANS=31H  THEN END
70    IF ANS=32H THEN PRINT "WAIT FOR COMMAND FROM EX..." : CALL16
80    GOTO 50
```

The switches on the DIP switch on the front panel marked * are set as shown below.



By turning on power to the ASCII module, this program will be executed automatically. The number of the BASIC-52 programs presently stored in the ASCII module will be displayed on the console device connected to CH1. Select between the BASIC-52 operating mode and interface mode with the EX CPU.

6.7 Program Saving/Loading

This section describes saving of BASIC-52 programs stored in the EEPROM of the ASCII module in a floppy disk (for program back-up) and transferring programs saved on a floppy disk to the ASCII module.

An example of a program to transfer BASIC-52 programs by the BASIC programs on the personal computer side and to save in a floppy disk or to load from a floppy disk is shown in the next and subsequent pages.

This example uses a T-3100 series lap-top personal computer as a personal computer and BASIC which activates in the T-3100 series. Implant when using with a personel computer of other model.

- Backing up BASIC-52 programs of ASCII module

An example of saving a BASIC-52 program stored in the EEPROM of the ASCII module in a file named PROM. TXT on the floppy disk of drive [A] is described.

```
10  ' Save BASIC-52 programs in ASCII/BASIC module into floppy disk
20  ' file name = SAVE.BAS
30  CLS:LOCATE  , ,1,7,8
40  DIM P$(1000)
50  OPEN "COM1:1200,N,8,1" AS #1
60  OPEN "A:PROM.TXT" FOR OUTPUT AS #2
70  '-------------------- Decide which program data to save
80  PRINT #1,"?XBY(7408H)"
90   LINE INPUT#1,A$
100    LINE INPUT#1,A$
110 A=VAL(A$)
120 PRINT CHR$(12)
130 PRINT "TOTAL PROGRAM COUNTS : ";A
140 INPUT "ENTER PROM NO.TO SAVE ",N
150 PRINT #1,"PROM";N;CHR$(&HD);
160   LINE INPUT#1,A$
170 IF RIGHT$(A$,5)<>"READY"  THEN 160
180 '-------------------- Receive program data from ASCII module
190 PRINT "Now getting BASIC-52 program data..."
200 PRINT:PRINT "PROGRAM No.";N
210 J=1
220 PRINT #1,"LIST";CHR$(&HD)
230 LINE INPUT#1,A$
240   P$(J)=A$:J=J+1
250   D$=RIGHT$(A$,5)
260 IF D$<>"READY" THEN 230
270 '-------------------- Display program data on terminal & save it i
nto floppy disk
280 FOR K=1 TO J-1
290   PRINT P$(K);:PRINT# 2,P$(K)
300 NEXT K
310 PRINT
320 CLOSE
330 END
```

NOTE

Precautions when executing this program are:

- Set the baud rate of CH1 on the ASCII module side to 1200 bps.
- This program sets the name of the backup file in line 60 and the No. of BASIC-52 program to save in line 140. Therefore, execute the program after changing the backup file name of line 60 by the BASIC-52 program to be saved when saving several BASIC-52 programs on a floppy disk.

- Transferring the backup program to the ASCII module

A program example of transferring a BASIC-52 program stored in a file PROM.TXT on the floppy disk of drive [A] to the RAM of the ASCII module is shown below.

```
10  ' Load BASIC-52 program in floppy disk into ASCII module
20  ' file name = LOAD.BAS
30  CLS:LOCATE  , ,1,7,8
40  DIM P$(1000)
50  OPEN "COM1:1200,N,8,1" AS #1
60  OPEN "A:PROM.TXT" FOR INPUT AS #2
70  '-------------------- Load program data from floppy disk
80  PRINT "Now getting BASIC-52 program data..."
90  J=1
100 IF EOF(2) THEN 140
110   LINE INPUT #2, B$
120   P$(J)=B$:J=J+1
130 GOTO 100
140 '-------------------- Transmit program data to ASCII module
150 PRINT #1,"RAM";CHR$(&HD);
160   LINE INPUT#1,A$
170 IF RIGHT$(A$,5)<>"READY"  THEN 160
180 FOR K=2 TO J-1
190   L=LEN(P$(K))
200   FOR M=1 TO L
210     PDATF$=MID$(P$(K),M,1)
220     PRINT #1,PDATF$;
230     PDATA$=INPUT$(1,#1)
240     IF PDATA$<>PDATF$ THEN 230
250   NEXT M
260   PRINT #1,CHR$(&HD);:PRINT #1,CHR$(&HA)
270   DUMA$=INPUT$(1,#1)
280   IF DUMA$<>CHR$(&H3E) THEN 270
290   PRINT P$(K)
300 NEXT K
310 PRINT
320 CLOSE
330 END
```

NOTE

Precautions when executing this program are described below:

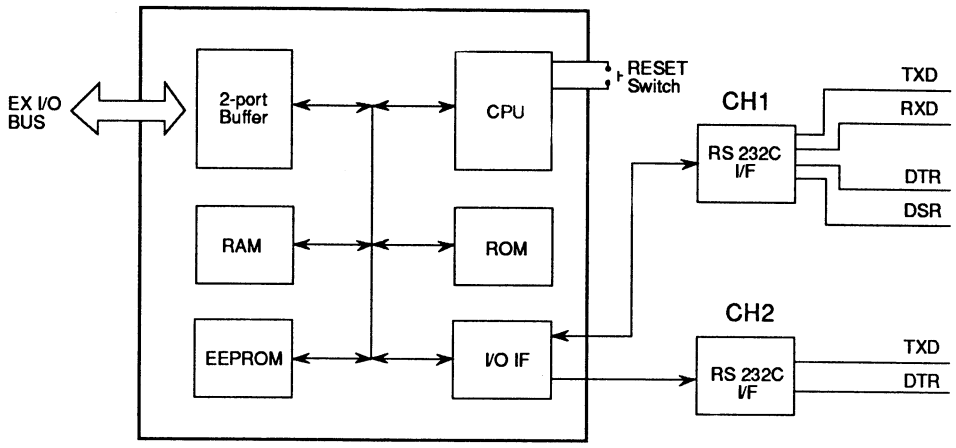
- Set the baud rate of CH1 on the ASCIi module side to 1200 bps.
- After executing this program, the BASIC-52 program will be located in the RAM. SAVE the program in the EEPROM when interfacing with the EX CPU afterward.

Item			Specification	Remarks
Power Supply	Voltage		5VDC ±5%	Supplied by EX CPU
	Current Consumption		Max. 0.8A	
	Holding Time (Momentary power failure)		Normal operation within 10 msec (PC unit)	
Environment	Temperature			
		Operating	0° to 55°C (32° to 131°F)	
		Storage	-20° to 75°C (-4° to 167°F)	
	Humidity		20~90% RH	No condensation
	Vibration		16.7 Hz, 3 mm P-P	Current not fed
	Shock		10G 3 times in X,Y, and Z directions	Current not fed
	Noise Immunity		1000Vp-p 1 μs Resistance (Conforms to NEMA ICS3-304)	
	Insulation		None	RS-232C port-internal logic parts
	Atmosphere		No corrosive or flammable gas	
	Dust Density		10 mg/m^3 or less	
Withstand Voltage	Power Supply		1500 Vac (1 min)	
Cooling			Natural air cooling	

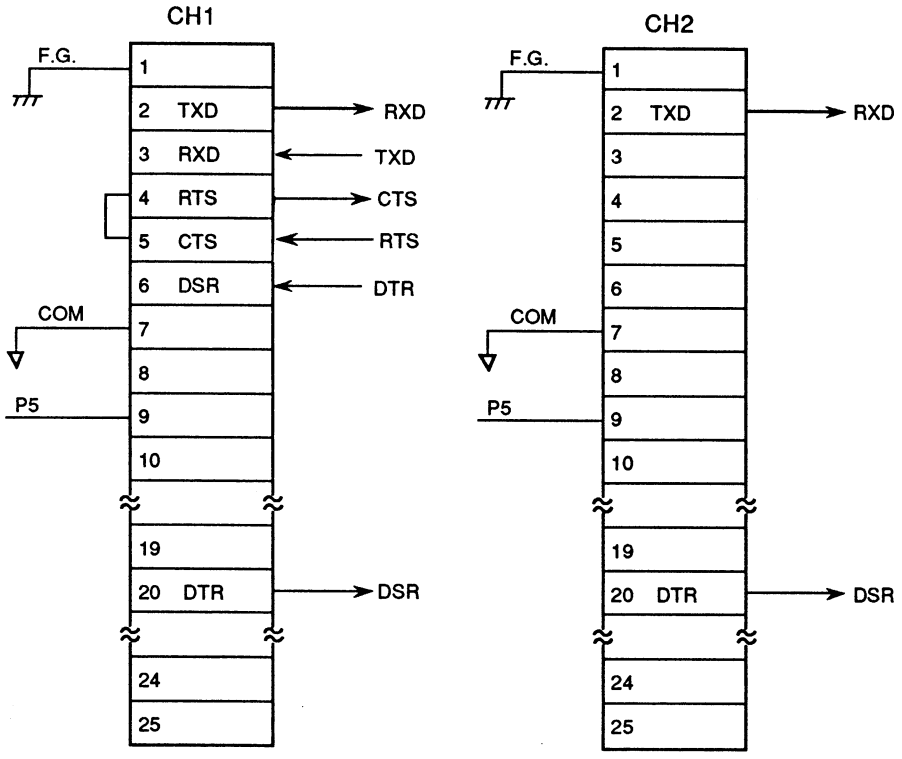
Item	Specification	Remarks
Language	Intel MCS BASIC-52	
Memory Capacity	For data/programs 29 Kbytes Buffer area for interface with EX CPU 2 Kbytes For program storage 32/64 Kbytes	
Function	• Input and output with ASCII peripheral devices • Arithmetic processing by BASIC-52: Substitution, four arithmetic operations, logical operations, trigonometric functions, exponential function, logarithmic function, absolute value, square root, and others • Interface utilities with EX CPU 18 types of built-in subroutines	
Operating System	Asynchronous independent operation (Synchronization by start instruction through interrupt by EX CPU (READ/WRITE command) and completion output)	
Transmission Interface	2 channels conforming to RS-232C CH1: For input and output CH2: Dedicated for output	
Transmission System	Start-stop synchronization (asynchronous)	
Transmission Format	Start bit 1 bit Data 8 bits Parity None Stop bit 1 bit	

Item	Specification	Remarks
Transmission Speed	CH1: 110, 300, 600, 1200, 2400, 4800, 9600, 19200 bps CH2: Set by BAUD statement of BASIC-52	Set by DIP switch on front panel.
Connected Device	Printer, CRT, Keyboard, and other RS-232C interface peripheral devices	
EX I/O Interface	Option module or X+Y 4W	Selection by internal jumper setting.
Data Exchange Method	By READ/WRITE command or READ/ WRITE sequence of EX CPU	
RAS Functions	Self-diagnosis (at initialization) watchdog timer (200 msec) (CPU reset if error occurs.)	
Indication	RUN: Normal run indication TX1 : CH1 transmission data RX1 : CH1 reception data TX2 : CH2 transmission data	

Circuit Configuration:



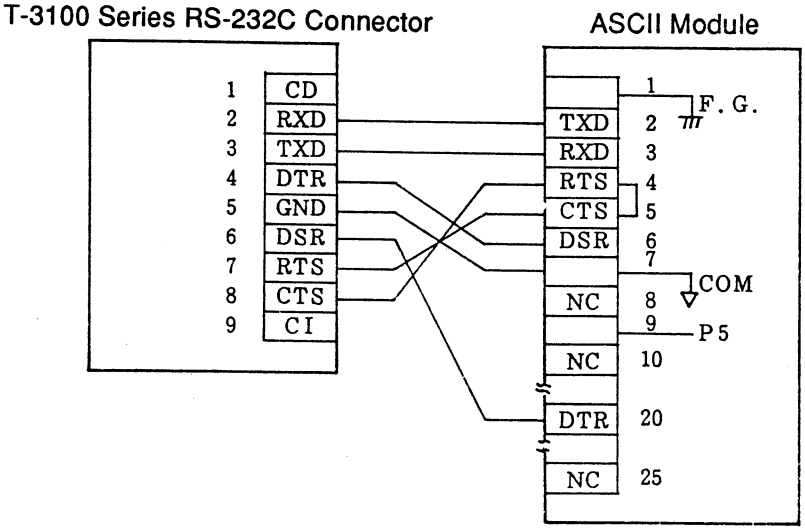
Connector Connection:



* DTR is always output while the power to CH1 and CH2 is on.

An example of connecting a personal computer to the ASCII module is shown.

(1) T-3100 Series - CH1 of ASCII Module



4. Error Message and Troubleshooting Lists

This appendix lists error messages detected by the ASCII module and describes the troubleshooting of them.

The errors that occur with the ASCII module can be divided into three types:

(1) Errors during start-up of the ASCII module.

(2) Errors during the LOCAL mode of the ASCII module for editing and debugging of the BASIC-52 programs.

(3) Errors during interfacing of the ASCII module and EX CPU.

These three types of errors and indicated message/error description/troubleshooting are described in the following pages.

4.1 Errors During Start-up of ASCII Module

LED Indication	Error Description	Troubleshooting
○ RUN ○ TX1 ○ TX2 ○ RX1	Power supply abnormality	Is required power (5Vdc) supplied to the ASCII module? Check wiring, connector connection, and other pertinent items.

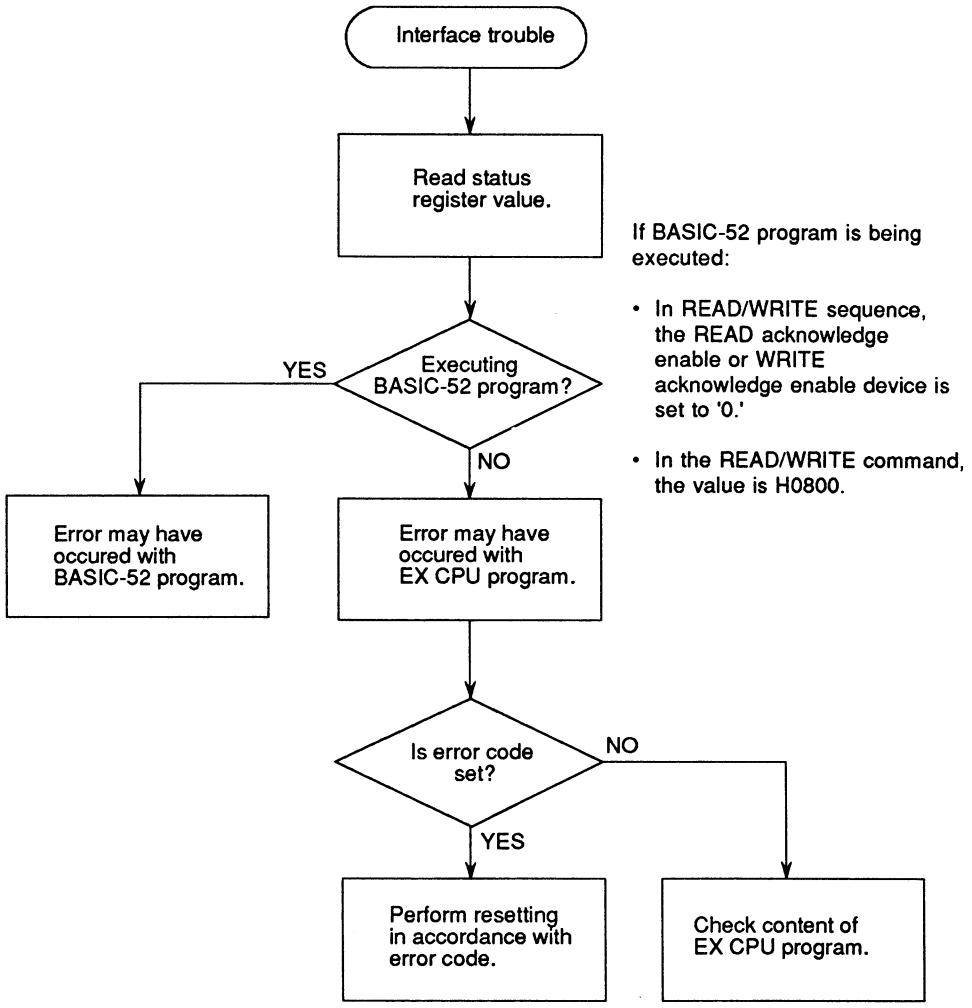
NOTE: ● LIT
☼ BLINK
○ OFF

4.2 Errors During BASIC-52 Program Editing and Debugging

Refer to 5.5 List of BASIC-52 Error Messages in APPENDIX 5. BASIC-52 LANGUAGE in the Appendix.

4.3 Interfacing Error Between ASCII Module and EX CPU

Check the EX CPU and ASCII module programs using the following troubleshooting flowchart as a reference if anticipated processing cannot be executed between the EX CPU and ASCII module during interfacing of the ASCII module and EX CPU.



Troubleshooting of errors is described below.

• BASIC-52 program error

Troubleshoot as follows.

(1) Connect the console device to the ASCII module and execute the program.

(2) Refer to APPENDIX 5.5 List of BASIC-52 Error Messages and check the error nature if the BASIC-52 error message is displayed on the console device and the execution of the BASIC-52 program is interrupted.

(3) Correct the BASIC-52 program.

- EX CPU program error (If error code is not set)

Check the program content of the EX CPU. The following items are particularly important:

- Is the timing to set/reset each device of the command registers correct in the READ/WRITE sequence mode?

- Is holding of condition input correctly performed during the execution of the READ/WRITE command?

- Error of EX CPU program (if error code is set)

If an error code is stored in the status register, take an action in accordance with the following table. Also read sections 5.1.1 Register Assignment (READ/WRITE Sequence) and 5.2.2 READ/WRITE Commands in CHAPTER 5 BASIC READ AND WRITE OPERATIONS.

The mark * in the chart shows error codes that are set only when the READ/WRITE command is used.

Error Code	Error	Troubleshooting
01H*	ASCII module is not set in designated channel number.	Check I/O allocation of EX CPU.
02H*	Hardware error has occurred with ASCII module of the designated channel number, and accessing by the EX CPU is not possible.	The ASCII module has failed. Contact our service department or agent.
03H*	ASCII module of the designated channel number was not in an IDLE state during READ/WRITE command execution.	Check DIP switch settings on ASCII module and allow reception of an EX CPU command. Check whether or not EX CPU is processing to reset ASCII module.

Error Code	Error	Troubleshooting
04H*	An I/O NO SYNCHRO error has occurred in ASCII module of the designated channel number during READ/WRITE command execution.	Check that the ASCII module is correctly installed.
05H*	ASCII module was set to READ/WRITE command acknowledge enable state before READ/WRITE command execution finished.	Check whether or not EX CPU is processing to reset ASCII module.
06H	CPU error has been detected with the ASCII module of the designated channel number.	The ASCII module has failed. Contact our service department or agent.
07H	RAM error has been detected with the ASCII module of the designated channel number.	
08H	ROM error has been detected with the ASCII module of the designated channel number.	
09H	Watchdog timer error by watchdog timer set by the user has occurred with the ASCII module of the designated channel number.	Check whether or not operation other than that specified is performed with IE register during BASIC-52 program operation by ASCII module.
0AH	BASIC-52 program No. sent by EX CPU to applicable ASCII module was illegal.	Check whether or not illegal program No. (No. not meeting condition of $0 \leq n \leq$ [number of existing programs]; n: natural number) is sent to the ASCII module.

Error Code	Error	Troubleshooting
0BH	Command sent to the ASCII module of the designated channel number by EX CPU was illegal. Or, EX CPU set the READ start and data READ end devices simultaneously (when READ/WRITE sequence is used).	Check whether or not illegal command has been sent to ASCII module. If the READ/WRITE sequence is used, also check whether or not READ start and data READ end devices are set simultaneously.
0CH	The ASCII module of the designated channel number was executing the RESET routine.	Change the program so that READ/WRITE requests are not activated while ASCII module is executing the initialization routine.
82H*	Table size set in READ/WRITE command was outside of effective range.	Only table size matching EX CPU to be used can be set during EX CPU programming using GP, HP, or other hardware. Check is made so that set table size does not surpass effecrive register region. Normally, these two errors do not occur. If the two errors occur, the EX CPU program may have been destroyed. Check the program content.
83H*	Table size set in READ/WRITE command surpassed the effective register area.	
84H*	Basic-52 program No. or command sent to ASCII module of designated channel number was outside of effective range.	Check whether program No./command sent to ASCII module is inside the effective range. Effective numerical range is 1 to 511.
85H*	Program Nos. designated in READ and WRITE commands during simultaneous READ/WRITE execution were not identical.	Make program Nos. designated in READ and WRITE commands identical.

Error Code	Error	Troubleshooting
86H*	Set data word value surpassed the effective range inside the table size during WRITE process.	For example, this error code is set if 20 pieces of data are written even if table size is 10. Change the data word value/table size value.

5. Description of BASIC-52 Language

This chapter describes the BASIC-52 language used in the ASCII module.

5.1 Outline of BASIC-52

The ASCII module contains a micro controller called 8052AH-BASIC as its CPU. This micro controller incorporates a BASIC interpreter called MCS® BASIC-52. The BASIC language used in the ASCII module is MCS BASIC-52 (hereinafter called BASIC-52). The ASCII modules have their own BASIC-52 utilities which enable efficient interfacing with the EX CPU and BASIC-52 program editing.

The features of the BASIC-52 utilities that are used in the ASCII module can be summarized as follows.

(1) Dedicated data type conversion routines

The dedicated data type conversion routines needed for data exchanges between the EX CPU and ASCII module are provided.

(2) Dedicated interface routines

Dedicated routines for flag processing needed for interface between the EX CPU and ASCII module are provided.

(3) Management of BASIC-52 programs inside EEPROM

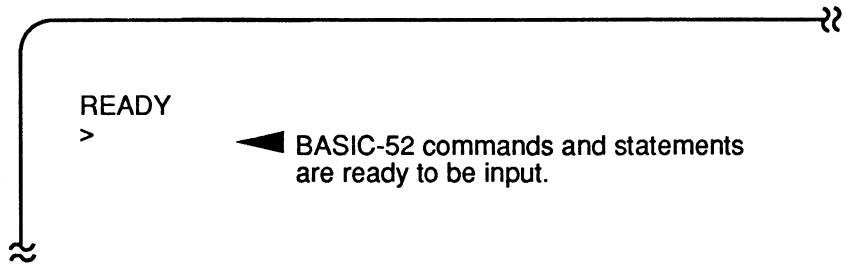
The ASCII module can save the BASIC-52 programs in the EEPROM and has routines dedicated for saving and managing these programs.

5.2 Basic Functions of BASIC-52

This section describes phrases used in BASIC-52, characters that can be used, and ranges of values that can be handled.

5.2.1 Operation modes of BASIC-52

BASIC-52 commands and statements can be input if 'READY' and the prompt '>' are indicated on the CRT screen when BASIC-52 is started.



Commands and statements are input in this state to operate BASIC-52, which has two operating modes. These are the direct and program modes.

- **Direct mode**

  Inputting a command in accordance with the BASIC-52 grammar without putting a line number, causes the command to be executed immediately after input. This operating mode is called the direct mode.

- **Program mode**

  Inputting a BASIC-52 statement after a line number between 0 and 65535 stores the statement in the memory as part of a program together with the line number. Once stored, the lines are executed by the RUN command and by the GOTO or GOSUB statement. This operating mode is called the program mode.

```
READY
>FOR I=1 TO 5: PRINT I,: NEXT
 1  2  3  4  5
>
```

<Direct Mode>

```
READY
>10 FOR I=1 TO 5: PRINT I,: NEXT
>20 END
>RUN
 1  2  3  4  5
READY
>
```

<Program Mode>

Of the commands and statements used in BASIC-52, commands can be used only in the direct mode. Some statements can be used in both the direct and program modes, while others can be used only in the program mode. This information is given in section 5.3 of this appendix. Descriptions of individual commands and statements are provided in section 5.3.

5.2.2 Character set and reserved words

BASIC-52 can use the following characters:

Alphabet letters (A ~ Z, a ~ z)
Numerals (0 ~ 9)
Special characteristics (!, @, $, #, ...)

- **Use of special symbols**

  Some of the special characters used in BASIC-52 have special meanings. These characters and their meanings are summarized below.

1. Hyphen (-)
   A hyphen is used when designating the line range by the LIST command (line ___ to line ___).

   Example: LIST 20 - 50

2. Colon (:)

   The colon is used as a statement delimiter in a multi-statement line.

   Example: A = B**2: PRINT "A=", A: END

3. Comma (,)

   If parameters are put in parallel in PRINT, INPUT, and other statements, a comma is used to delimit them.

   Example: INPUT A, B
   PRINT "A=", A, "B=", B

4. Question Mark (?)

   A question mark can substitute for a PRINT statement.

   Example: ? "ASC-6210"

5. Sharp (#)

   By putting a sharp after an output statement (PRINT or LIST statement), the output destination will be a serial port (printer, display, or other device).

   Example: LIST# 100
   PRINT# DATA1, DATA2

6. Space ( )

   As a rule, spaces are ignored in BASIC-52.

7. Alphabet letter 'H' (H)

   PH0, PH1, and other characters are used as a variation in PRINT statements of BASIC-52. 'H' after 'P' in these statements indicates printing out of numerical values in hexadecimal format.

   Example: PH0. XBY (7400H)
   PH1.# A

- **Reserved words**

Key words such as commands, statements, and function names used in BASIC-52 are called reserved words.
The reserved words cannot be used as variables.

The reserved words used in BASIC-52 are listed below.

List of BASIC-52 Reservation Words

ABS	FREE	.OR.	SGN
.AND.	GET	PCON	SIN
ASC	GOSUB	PI	SPC
ATN	GOTO	PH0.	SQR
BAUD	IE	PH1.	STEP
CALL	IF	PH0.#	ST@
CBY	INPUT	PH1.#	STRING
CHR	INS	POP	TAB
CLEAR	INT	PORT1	TAN
CLEARI	IP	PRINT	TCON
CLEARS	KILL	PRINT#	T2CON
CLOCK0	LD@	PROG	THEN
CLOCK1	LET	PUSH	TIME
CONT	LIST	RAM	TIMER0
COS	LIST#	RCAP2	TIMER1
CR	LOG	READ	TIMER2
DATA	MTOP	REM	TMOD
DBY	NEW	RENUM	TO
DIM	NEXT	RESTORE	UNTIL
DO	NOT	RETI	USING
ELSE	NULL	RETURN	WHILE
END	ON	RND	XBY
EXP	ONERR	RUN	XFER
FOR	ONTIME	SAVE	XTAL

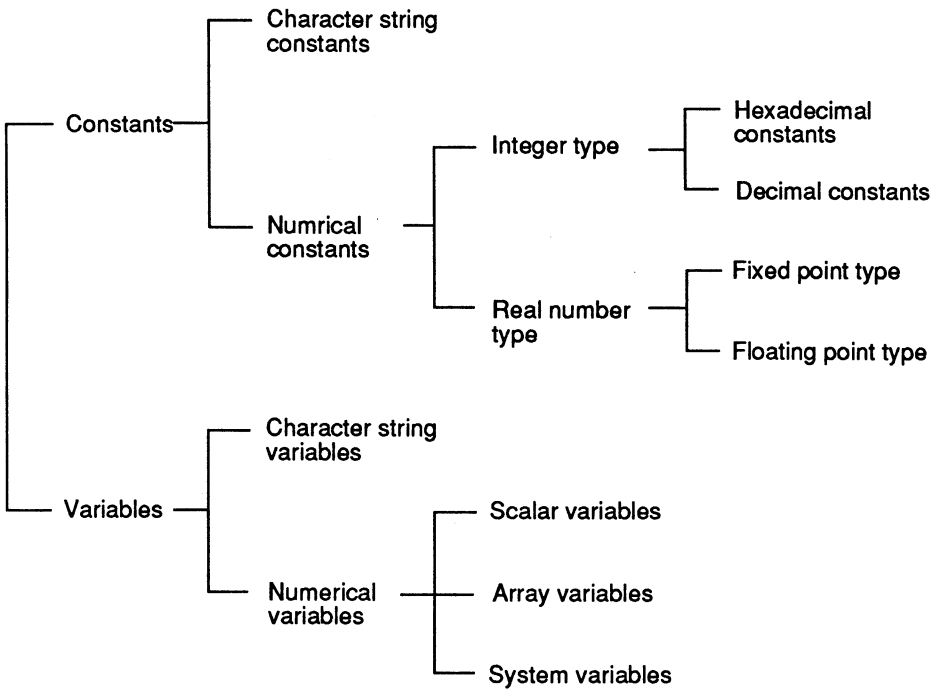
5.2.3 Line format

The rules concerning the line format used in BASIC-52 are as follows:

- The effective range of line numbers used in lines of programs is integers--- 0 ---- 65535.
- Programs are executed in the order of the line numbers. Adding or editing lines in the order of line numbers is not required.
- The maximum number of characters that can be input in one line is 79 characters.
- Several BASIC statements can be placed in one line if delimited by colons (:).

5.2.4 Constants and variables

Constants and variables used in BASIC-52 are summarized in the diagram below.



The range of values that can be handled by BASIC-52 is limited to:

±1E - 127 ~ ±.99999999E + 127

Significant figures are up to 8 digits in single precision.

• **Constants**

Constants include character string and numerical constants.

1. Character string constants

Alphanumerics, special symbols, and other characters enclosed in double quotations (") put in a string (up to the maximum number of characters that can be put in one line of a BASIC-52 program) are called character string constants.

Example: "ASCII/BASIC module"
"88-07-07 Thur."
"Target ? "

2. Numerical constants

The numerical constants used with BASIC-52 can be either integer or real-number types.

(1) Integer type

Any integer from 0 to 65535 (0FFFFH) can be used. Input and display by decimal and hexadecimal formats are possible. Place the suffix 'H' after a numeral to express a numerical value hexadecimally. Add the prefix '0' for numerals beginning with A through F in hexadecimal notation.

Example: 124
710H
0ABCDH

HINT If a numeral is expressed hexadecimally, it will be regarded as a numerical value and not as a character string.

(2) Real-number type

Any real numbers from ±1E - 127 to ±.99999999E + 127 can be used. Needless to say, integer-type numerical values are included. Input and display are performed for fixed and floating-point types.

Example: 19.63
-710.38
3.14 E + 3
-1.2345679 E + 9

- **Variables**

Variables are names consisting of alphanumeric characters matched in an area to store values used in BASIC-52 programs. Variable values are defined by the programs; they can be used for arithmetic operations, referencing, and other purposes.

Variables are divided into character string and numerical variables.

Example: $(0) = "CURRENT STATUS"
LINE_A = 1000

A '0' is substituted for numerical variables and a null string (string with no characters) is substituted for character string variables if reference is made before values are allocated to the variables.

1. Variable name setting

   The following rules apply when setting variable names with BASIC-52. The variable:

- Must start with an alphabet letter and must consist of alphanumerics and an underline (_) only.

- Must be less than 8 characters.

- Must not use a reserved word as the variable name or as part of the variable name.

<Example of correct variable name>

VAL1, ARRAY(0), LINE_B (254), $(1)

<Example of incorrect variable name>

DATA, CONST, VARTOP

2. Character string variables

   When character strings are handled during the execution of a BASIC-52 program, character string storage areas are secured as arrays by STRING statements. Individual character string variables are substituted or set in these arrays.

```
Example: 10      STRING 200, 20
         20      $(0) = "*** LINE CONTROL ***"
                  .          .
                  .          .
                  .          .
         100     INPUT "NAME? ", $(1)
                  .          .
                  .          .
                  .          .
```

3. Numerical variables

Numerical variables can be scalar, array or system variables.

(1) Scalar variables

Scalar variables are variables that are defined by the user and that have no dimension.

Example: A, R1, COUNT_1

(2) Array variables

Only linear arrays can be used with BASIC-52. The maximum subscript size is 254. Set the number of elements using the DIM statement if array variables are used.

Example:

```
10    DIM     A(100)
 .     .
 .     .
 .     .
100   A(0) = 1
 .     .
 .     .
 .     .
```

```
10    A1(0) = 5          ◄ The subscript size automatically becomes
 .     .                    10 if the DIM statement is omitted.
 .     .
100   A1(10) = 100
```

Array variable subscripts begin with 0, so the actual number of elements in each array will be the subscript number + 1.

Example: DIM SUM (254) ◄ Number of elements: 255

(3) System variables

BASIC-52 itself makes use of dedicated system variables. These variables cannot be used by the user as ordinary variables.

BASIC-52 uses the following system variables:

FREE: Shows how many bytes are empty in the ASCII module RAM. Substitution is not possible.

LEN: Shows the length of the BASIC-52 program presently selected. It cannot be substituted.

MTOP: Shows the last address of the area allocated for the BASIC-52 programs in the ASCII module RAM. Substitution is not possible.

5.2.5 Operators and functions

Arithmetic operations of BASIC-52 can be grouped into five types:

1. Mathematical operations
2. Relational operations
3. Logical operations
4. Functions
5. Character string operations

1. Mathematical operations

The mathematical operators include the following.

Operator	Operation	Execution Example
**	Exponential operation	> PRINT 2** 4 16
_	Negative sign	> A = 2 : B = -A : PRINT A, B 2 -2
*, /	Multiplication, division	> PRINT 2*3, 100/3 6 33.333333
+, -	Addition, subtraction	> PRINT 2 + 3, 2 - 3 5 -1

- Exponential operation exponents are 255 maximum.

2. Relational operations

Relational operators are used when two numerical values are compared. If the result of a comparison is true, '65535 (0FFFFH)' will be returned. If false, '0' will be returned. Such results are utilized to change the program flow in conditional judgment or other statements.

Operator	Operation	Execution Example
=	Equal to	> A = 1 : B = 0 : PRINT A = B 0
< >	Not equal to	> A = 1 : B = 0 : PRINT A< > B 65535
<	Less than	> A = 1 : B = 0 : PRINT A < B 0
>	Greater than	> A = 1 : B = 0 : PRINT A > B 65535
<=	Less than or equal to	> A = 1 : B = 0 : PRINT A <= B 0
>=	Greater than or equal to	> A = 1 : B = 0 : PRINT A >= B 65535

Examples of use in IF statements are shown below:

```
200  IF  C1 = 100  THEN  1000
 :    :

100  IF  A + B < > 0  THEN  A1 = A1 + 1 :  B1 = B1 + 1
 :    :
```

3. Logical operations

Logical operators are used when several conditions are checked or when bit or Boolean operations are performed. Conditional numerical values must be integers from 0 to 65535 (0FFFFH). A BAD ARGUMENT error results if values are not within the range 0 to 65535. Numerals following a decimal point will be truncated if a non-integer is used. A '1' is given for each bit if operation results are true; '0' is given if false.

As with relational operators, logical operators can be used to change the program flow. Logical operators can also check data, matching it to bit patterns.

Examples of use of logical operators and applications are listed in the following page.

Operator	Operation	Execution Example
AND.	AND	> A = 1 : B = 0 : PRINT A. AND .B 0
.OR.	OR	> A = 1 : B = 0 : PRINT A. OR .B 1
.XOR.	Exclusive OR	> A = 1 : B = 0 : PRINT A. XOR. B 1
NOT ( )	Negation	> A = 1 : B = 0 : PRINT NOT (A),NOT (B) 65534 65535

```
> PRINT 5. AND. 2
  0
```
$5 = (101)_2$, $2 = (010)_2$
therefore 5. AND. 2 = $(000)_2$ = 0

```
> PRINT 3. OR. 8
  11
```
$3 = (0011)_2$, $8 = (1000)_2$
therefore 3. OR. 8 = $(1011)_2$ = 11

```
> PRINT 65535 .XOR. 0
  65535
```
65535 = $(1111\ 1111\ 1111\ 1111)_2$
0 = $(0000\ 0000\ 0000\ 0000)_2$
therefore 65535 . X OR.
0 = $(1111\ 1111\ 1111\ 1111)_2$
= 65535

```
> A = 10 : PRINT A, NOT(A), - (A+1), NOT(NOT(A)), 65535 - A
 10   65525   -11   10   65525
```

```
10  INPUT "A, B, C ?", A, B, C
20  IF  A=B    .AND.  B=C    THEN 500
30  IF  A<B    .OR.   B<C    THEN 1000
40  IF  NOT(A<0) .AND. NOT (B<0). AND. NOT (C<0) THEN 1500
```

4. Functions

Functions perform predetermined operations with given arguments and return various operation results as values.

BASIC-52 has numerical functions such as SIN( ) (sine) and SQR( ) (square root), special functions such as XBY( ) and TIME, and functions such as MTOP and FREE that provide system variables.

The individual functions will be described in detail. The working functions of several functions and operational precautions are explained below.

The BASIC-52 programs have four functions regarding output formatting during output, namely, CR, SPC, TAB and USING functions. These functions can be used in PRINT, PRINT #, PH0, PH1, PH0.# and PH1.# statements.

The usage of special functions requires attention. The special functions directly handle special function registers of 8052AH. Refer to the operation manual of 8052H for the details of the operation of the spedial function registers.

Basically, all the special functions can be set in addition to value reading.

However, setting of values using these special functions without well understanding the 8052HAH and ASCII module functions may result in misoperation of the ASCII modules and should never be performed.

5. Character string operations

BASIC-52 has two functions related to character string operations, namely, ASC( ) and CHR( ). Detailed descriptions will be given later; a simple use example is given below.

```
>LIST
10          STRING 100,20
20        $(0)="ASCII/BASIC module  "
30        $(1)="ASC-6210"
40        PRINT $(0), $(1)
50        FOR I=1 TO 5
60        PRINT ASC($(0), I), CHR($(0), I)
70        NEXT I

READY
>RUN

ASCII/BASIC module  ASC-6210
 65 A
 83 S
 67 C
 73  I
 73  I

READY
>
```

6. Priority order of operations

Operations of BASIC-52 are performed in the following priority order.

1. Those in parentheses ( )
2. Functions (including NOT( ))
3. Exponents (power method) **
4. Negative sign (-)
5. Multiplication, division *, /
6. Addition, subtraction +, -
7. Relational operators =, < >, <, >, <=, >=
8. Logical product .AND.
9. Logical sum .OR.
10. Exclusive logical sum .XOR.

5.2.6 BASIC-52 stack structure

Three stacks necessary for operation of BASIC-52 are described below.

- **Argument Stack (A-STACK)**

  The A-STACK is used for data exchanges between BASIC-52 and built-in (CALL) subroutines. Use the PUSH/POP statements of BASIC-52 for data exchanges. Each value put in the A-STACK occupies 6 bytes. The A-STACK area is 210 bytes; the number of data items to be pushed at one time must be limited to 30 as BASIC-52 has to process programs also. An A-STACK error results if more data is pushed or, conversely, if a POP statement is used although data is not pushed.

- **Control Stack (C-STACK)**

  This stack stores information regarding loop processing (DO-UNTIL, DO-WHILE, and FOR-NEXT statements) and subroutine processing (GOSUB-RETURN) in a BASIC-52 program. If a C-STACK error results during BASIC-52 program execution, check that the processing has been performed properly.

- **Internal Stack (I-STACK)**

  This is the information area to decide if mathematical expressions are correct.

  Only the A-STACK is actually used during BASIC-52 operations.

5.2.7 Glossary and notation method

The following parameters, terms, and notation methods are used with the commands of BASIC-52 described in section 5.3 and the subsequent sections.

- [CR] This shows input of the Carriage Return key (Enter key) on the console device keyboard.

  Example: RUN [CR]

- [Control-C] Shows that the control key (CTRL or Ctrl Key) and "C" key are continuously input by the console device key board.

- <Integer> This notation indicates that the effective range of a parameter required by a BASIC-52 command is an integer.

  Example: PROM <integer> [CR]

- <Line No.> This notation indicates that the parameter required by a BASIC-52 command is a program line No.

  Example: LIST <line No.>-<line No.>

- <Variable> This notation indicates that the parameter required by a BASIC-52 command is a variable name.

  Example: POP <variable>

- <Mathematical Expression>

  This notation indicates that the parameter required by a BASIC-52 command is expressed by a mathematical expression that contains an operator, constant, and variable.

  Examples of a mathematical expression are :
  4*A
  (EXP(I) + EXP (-I))/2

  Example: SQR (<mathematical expression>)

- <Relational Expression>

  This notation indicates that the parameter required by a BASIC-52 command is an expression that contains a relational operator.

  Example: DO - UNTIL <relational ex pression>

- ..... This shows that an item is repeated.

  Example: ON <mathematical expression>
  GOTO <line No.>, ...

An item put in < > must be designated in a command.
The command can be used even if an item put in [ ] is not designated.
The < > and [ ] are descriptive symbols and should not be input during processing.

5.3 Commands, Statements, and Functions

The commands, statements, functions, and system variables of BASIC-52 used in the ASCII module are described in the following pages.

First, the commands, statements, functions, and system variables are listed. A detailed description of each individual item follows. Data type conversion special routines, interface special flag processing routines and other items will be described separately.

Lists of Commands, Statements, and Functions

• Commands

Program Editing and Execution Commands

Command	Function	Use Example	Reference Page
NEW	Deletes a program in the RAM and clears all variables.	NEW	214
RUN	Executes a program. (Clears all variables before execution.)	RUN	233
CONT	Resumes execution of a program whose execution has been interrupted by the Control-C keys or by a STOP statement.	CONT	180
LIST	Outputs a program to the equipment connected to CH1 of the ASCII module.	LIST LIST 10 - 100	210
LIST#	Outputs a program to the equipment connected to CH2 of the ASCII module.	LIST#	211
NULL	Sets the number of NULL characters (00H) to be output after a carriage return.	NULL NULL 100	216
RENUM	Renumbers line Nos. of a program in the RAM.	RENUM RENUM 500, 10	231

Program File Storage and Management Commands

Command	Function	Use Example	Reference Page
PROM	Selects the program of the designated No. from programs saved in the EEPROM and sets the BASIC-52 to the EEPROM mode.	PROM 7	226
RAM	Sets the BASIC-52 to the RAM mode.	RAM	228
XFER	Transfers the program presently selected to the RAM and sets the BASIC-52 to the RAM mode.	XFER	251
SAVE	Saves a program in the RAM to the EEPROM.	SAVE	234
INS	Saves a program in the RAM to a position of the EEPROM designated by the program name.	INS 3	201
KILL	Deletes a program saved inposition of the EEPROM designated by the program name.	KILL 6	204

• **Statements**

Command	Function	Use Example	Reference Page
BAUD	Sets baud rate for the ASCII module CH2 port.	BAUD 300	172
CALL	Calls ASSEMBLER subroutines.	CALL 10 CALL 0A000H	173
CLEAR	Initializes to '0' all variables and initializes interrupt and stack information activated by the BASIC-52.	CLEAR	175
CLEAR I	Clears all interrupts activated by BASIC-52.	CLEAR I	177
CLEAR S	Initializes all stacks used by BASIC-52.	CLEAR S	178
CLOCK 1	Switches on the real-time clock.	CLOCK 1	179
CLOCK 0	Switches off the real-time clock.	CLOCK 0	179
DATA	Sets numerical values read by READ statement.	DATA 10	183
READ	Reads numerical values set in DATA statements into variables.	READ A	183
RESTORE	Returns pointer indicating the sequence of numerical values in DATA statements to be read next by READ statement to the top of the DATA statements.	RESTORE	183
DIM	Designates sizes of array variable elements and allocates memory areas.	DIM A(100)	185
DO ~ UNTIL	Repetitively executes statements contained between the DO and UNTIL statements until conditions given in UNTIL statement are met.	DO ~ UNTIL A=10	186

Command	Function	Use Example	Reference Page
DO ~ WHILE	Repetitively executes statements between the DO and WHILE statements as long as the conditions given in WHILE statement are met.	DO ≀ WHILE A>0	186
END	Ends program execution and returns to the direct mode.	END	187
FOR ~ TO ~ (STEP)~ NEXT	Repetitively executes statements between the FOR and NEXT statements until the variable given in the FOR statement reaches the stated value.	FOR I=0 TO 10 STEP 2 ≀ NEXT I	189
GOSUB ≀ RETURN	Calls subroutines beginning with the line No. that follows GOSUB. Returns to the main program when the RETURN statement is executed during subroutines.	100 GOSUB 1000 ≀ ≀ 1000 REM SBRTN1 ≀ ≀ 1500 RETURN	193
GOTO	Unconditionally transfers program execution control to the line no. that follows GOTO.	GOTO 100	195
ON ~ GOSUB	Calls subroutines beginning with the line No. that follows GOSUB in accordance with expression value.	ON A GOSUB 100,200	217
ON ~ GOTO	Transfers program execution control to the line No. that follows GOTO in accordance with expression value.	ON A GOTO 100, 300	217
IF ~ THEN ~ ELSE	Decides conditions of relational expressions that follow the IF statement and changes the program flow.	IF A<B THEN A=0 ELSE A=100	197
INPUT	Reads numerical values, character strings, and other items from the console connected to CH1 of the ASCII module and substitutes them into variables.	INPUT A	199

Command	Function	Use Example	Reference Page
LET	Substitutes values into variables. (LET can be omitted)	LET A=10	209
ONERR	Enables the interrupt that processes errors when they occur. Transfers program execution control to the line No. that follows	ONERR 9000	218
ONTIME ≀ RETI	Interrupts if the real-time clock value becomes larger than the value that follows ONTIME. Transfers program execution control to the subroutine that starts with the line No. that follows the numerical value. Returns to the main program when this processing is finished during subroutine operation.	100 ONTIME 2, 200 ≀ ≀ 200 REM TIMER INTRPT ≀ ≀ 500 RETI	219
PRINT	Displays numerical values, variable data, character strings, results of mathematical expressions, and other items on the equipment connected to CH1 of the ASCII module. P. and ? (question mark) are short forms of the PRINT statement.	PRINT A P. "BASIC-52" ? x**2	224
PRINT#	Outputs numerical values, variable data, character strings, results of mathematical expressions, and other items to the equipment connected to CH2 of the ASCII module. P.# and ?# are short forms of the PRINT# statement.	PRINT# A P.# "BASIC-52" ?# x**2	225
PH0. PH1.	Displays numerical values, variable data, character strings, results of mathematical expressions, and other items on the equipment connected to CH1 of the ASCII module. Numerical values from 0 to 65535 (0FFFFH) are displayed hexadecimally. PH1. is always indicated in 4 hexadecimal digits. PH0. is displayed omitting '0' in the higher two digits if the numerical value is less than 255 (0FFH).	PH0. A PH1. A	221

Command	Function	Use Example	Reference Page
PH0.# PH1.#	Outputs numerical values, variable data, character strings, results of mathematical expressions, and other items on the equipment connected to CH2 of the ASCII module. Numerical values from 0 to 65535 (0FFFFH) are output hexadecimally. PH1 is always indicated in 4 hexadecimal digits. PH0. is output omitting '0' in the higher two digits if the numerical value is less than 255 (0FFH).	PH0.# A PH1.# A	222
CR	Moves the cursor to the top of the current line.	FOR I=1 TO 1000: P. I,CR,: NEXT I	182
SPC(n)	Outputs spaces of the quantity specified by n.	PRINT A, SPC(5), B	238
TAB(n)	Outputs spaces to the designated position in the line in which the cursor is positioned.	PRINT TAB(5), A	244
USING( ) U.	Outputs numerical values after converting into the designated format. U. is a short form of the USING function.	PRINT USING (F3), A ? USING(##.##), A	248
PUSH	Pushes mathematical expressions that follow PUSH to the argument stack (A-STACK).	PUSH 10*3, 0, A	227
POP	Substitutes numerical values popped from the argument stack for variables that follow POP.	POP A, B, C	223
REM	Adds comments in programs.	REM INPUT DATA	230
STOP	Interrupts program execution and returns to the direct mode.	200 STOP	240
STRING	Designates the lengths of character string variables and allocates memory areas.	STRING 100, 10	241

Command	Function	Use Example	Reference Page
ST@	Pops and stores values in the argument stack in the address designated by the mathematical expression that follows ST@.	ST@ 7300H	243
LD@	Pushes numerical values, which are stored in the address designated by the mathematical expression that follows LD@, to the argument stack.	LD@ 7300H	206

- **Functions and operators**

Mathematical Operations

Command	Function	Use Example
+	Performs addition.	1 + 2
-	Performs subtraction.	3 - 2
*	Performs multiplication.	2 * 3
/	Performs division.	4/2
**	Performs exponential operations.	3**5

Relational Operations

Command	Function	Use Example
=	Equal to	PRINT 1 = 0
< >	Not equal to	PRINT 1 < > 0
<	Less than	PRINT 1 < 0
>	Greater than	PRINT 1 > 0
< =	Less than or equal to	PRINT 1 < = 0
> =	Greater than or equal to	PRINT 1 > = 0

Logical Operations

Command	Function	Use Example
.AND.	Provides AND.	1. AND. 3
.OR.	Provides OR.	8. OR. 5
.XOR.	Provides exclusive OR.	6. XOR. 2

Numerical Processing

Command	Function	Use Example	Reference Page
NOT( )	Provides 1's complement.	NOT (0)	215
ABS( )	Provides absolute value. $\|x\|$	ABS (-4)	169
INT( )	Provides integer part of numerical value.	INT (3. 14)	203
SGN( )	Provides sign of numerical value.	SGN (0)	236
SQR( )	Provides square root value. $\sqrt{x}$	SQR (100)	239
RND	Provides random number between 0 and 1.	PRINT RND	232
EXP( )	Provides exponential function value. e^x	EXP (1)	188
LOG( )	Provides logarithmic function value. $\log_e x$, $\ln x$	LOG (2)	212
SIN( )	Provides sine value. $\sin x$	SIN (PI)	237
COS( )	Provides cosine value. $\cos x$	COS (0)	181
TAN( )	Provides tangent value. $\tan x$	TAN (PI/4)	245
ATN( )	Provides arctangent value. $\arctan x$	ATN (1)	171

Character String Processing

Command	Function	Use Example	Reference Page
ASC	Provides character code of a character in designated position of a character string.	ASC (A) ASC ($(0), 2)	170
CHR	Converts a character code into a character.	CHR (65) CHR ($(0), 3)	174

Special Functions

Command	Function	Use Example	Reference Page
DBY ( )	Reads data in and assigns value to the internal data memory of the ASCII module.	A=DBY(3FH) DBY(15H)=100	184
XBY ( )	Reads data in and assigns value to the external data memory of the ASCII module.	A=XBY(7400H) XBY(7400H)=0	250
GET	Reads key input from the console device connected to CH1 of the ASCII module and provides character codes if the key input is valid.	10 A=GET	192
IE	Assigns value to the register IE of 8052AH and activates the watchdog timer of the ASCII module.	IE=IE.OR.2	196
PORT1	Reads value at and assigns value to the P1 I/O port of 8052AH.	PH0. PORT1	224
TIME	Reads value in and assigns value to the real time clock in BASIC-52.	TIME=0	246
TIMER0	Reads values of TH0 and TL0 registers of 8052AH.	P. TIMER0	247
TIMER1	Reads values of registers TH1 and TL1 of 8052AH.	P. TIMER1	247
TIMER2	Reads values of registers TH2 and TL2 of 8052AH.	P. TIMER2	247
XTAL	Reads and sets frequency values used by BASIC-52.	P. XTAL	252
RCAP2	Reads and sets values of registers RCAP2H and RCAP2L of 8052AH.	RCAP2=65380	229

System Variables

Command	Function	Use Example	Reference Page
MTOP	Reads and sets the value of the last address of the RAM area allocated for BASIC-52 programs.	PRINT MTOP	213
LEN	Provides the length of BASIC-52 program presently selected.	PRINT LEN	208
FREE	Shows how many bytes are available as an empty area in the RAM.	PRINT FREE	191

Constants

Command	Function	Use Example	Reference Page
PI	Provides circle ratio (π) = 3.1415926.	S=X**2*PI	220

ABS (<Mathematical Expression>)

Function

FUNCTION — Provides the absolute value of the <mathematical expression> value.

Example:

```
>PRINT      ABS(-7),: PRINT ABS(7)
7      7
>PRINT      ABS(0)
0
```

ASC (<Character/Character String>, [Mathematical Expression])

Function

FUNCTION — Provides the character code of the position designated in a character string, or of a character.

DESCRIPTION — If a character is given, a character code for the character is given. If a character string is given, the character code of the position designated by the mathematical expression is given.

Example:

```
> PH0.    ASC(A)
  41              ◄ An ASCII code for character 'A' will be given.

>10       STRING 50, 10
>20       $(0) = "ABCDEFGH"
>30       FOR I = 1 TO 8
>40       PH0. ASC ($(0), I),
>50       NEXT I
>RUN

41H 42H 43H 44H 45H 46H 47H 48H ◄ An ASCII code for characters in a character string consisting of eight characters A to H will be given.

READY
>

>LIST
10        STRING 50, 10
20        $(0)="ABCDEFGH"
30        FOR I=8 TO 1 STEP -1
40        ASC($(1), (9-I))=ASC($(0), I)
50        NEXT I
60        PRINT $(0): PRINT $(1)
70        FOR I=0 TO 1
80        FOR J=1 TO 8
90        PRINT ASC($(I), J),
100       NEXT J
110       NEXT I

READY                ◄ Characters can be substituted in optional positions of optional character strings using the ASC( ) function.
>RUN

ABCDEFGH
HGFEDCBA

65 66 67 68 69 70 71 72 72 71 70 69 68 67 66 65
READY
>
```

ATN <(Mathematical Expression)>

Function

FUNCTION — Provides arctangent value (arctangent function). (y=arctan x)

DESCRIPTION — Provides arctangent value of <mathematical expression>. The unit of the arctangent value is radian. Calculations are performed in 7 digits of valid numerals. The range of values of this function is from $-\pi/2$ to $\pi/2$.

Example:

```
>

>LIST
10      PRINT USING(###.###)
15      PRINT "      DEG    RAD    SIN()   COS()  TAN()  ARCTAN()"
20      FOR DEG=-180 TO 180 STEP 60
30      RAD=DEG*PI/180
40      PRINT DEG,RAD,SIN(RAD),COS(RAD),TAN(RAD),ATN(RAD)
50      NEXT DEG

READY
>RUN

DEG       RAD       SIN()      COS()      TAN()      ARCTAN()
-180.000  -3.141     0.000     -1.000      0.000     -1.262
-120.000  -2.094    -0.866     -0.500      1.732     -1.125
-60.000   -1.047    -0.866      0.500     -1.732     -0.808
 0.000     0.000     0.000      1.000      0.000      0.000
 60.000    1.047     0.866      0.499      1.732      0.808
 120.000   2.094     0.866     -0.500     -1.732      1.125
 180.000   3.141     0.000     -1.000      0.000      1.262
READY
>
```

BAUD <Mathematical Expression>

Statement (Direct/Program)

FUNCTION — Sets the baud rate for CH2 in the ASCII module.

DESCRIPTION — Sets the baud rate for CH2 in the ASCII module.

The baud rate of CH2 is set to 0 if a baud rate is not set using this command. Be sure to set a baud rate using this command before using CH2.

The upper limit of the CH2 baud rate is 19200 bps. A baud rate can be set within this range to match the baud rate of the equipment to be connected to CH2. Normally, the baud rate is 110, 300, 600, 1200, 2400, 4800, 9600, or 19200 bps, or other speed.

NOTE

Two precautions are listed below for use during the execution of BASIC-52 statements using CH2 (PRINT #,PH0.# and PH1.#). The first precaution is to set a baud rate of 4800bps or less when the above statement is executed and data is output during interfacing with the EX CPU or during operation of the watchdog timer. The second precaution is to prevent input of commands from the EX CPU during data output through CH2.
Data output errors may be caused if these two precautions are not observed.

Example:

```
>BAUD 300
>PRINT# "ASCII/BASIC module"
 (Character strings are output to the equipment connected to CH2 at a
baud rate of 300 bps.)
READY
>
```

CALL <Integer>

Statement (Direct/Program)

FUNCTION

Calls an ASSEMBLER subroutine.

DESCRIPTION

If this statement is executed by designating an execution start address of an ASSEMBLER subroutine after CALL, processing of the ASSEMBLER subroutine is performed. Executing this statement after designating an effective No. beginning with 0 after CALL, the ASSEMBLER routines incorporated in the ASCII module are automatically called for processing. Refer to "5.4 Built-in Subroutines" for the details of built-in subroutines.

Example:

```
>N=1: PUSH N: CALL 17

ROM 1 PROGRAM
*** PROGRAM COUNTS CHECK ***
PROGRAM COUNTS = 8


READY
>

>CALL 0A000H
```

◄ Performs processing of CALL 17 of built-in subroutine.

◄ Calls the ASSEMBLER routine created by the user. (This routine starts at Address 0A000H.)

CHR (<Numerical Value/Character String>, [Mathematical Expression])

Function

FUNCTION — Provides characters that are expressed by values or characters designated in character strings.

DESCRIPTION — If a numerical value is given, the character expressed by that character code is given. If a character string is given, a character that is in the position designated by the mathematical expression is provided.

Example:

```
>PH0. CHR(41H)
 A

>10       STRING 50, 10
>20       $(0)="ABCDEFGH"
>30       FOR I=1 TO 8
>40       PH0. ASC($(0), I), SPC(2), CHR($(0), I)
>50       NEXT I
>RUN

41H       A
42H       B
43H       C
44H       D
45H       E
46H       F
47H       G
48H       H

READY
>
```

NOTE CHR ( ) cannot substitute a character substitution statement. Neither, it can be used to compare character strings such as in IF statements. Both of these uses result in BAD SYNTAX errors.

```
CHR($(0), 1) = H
IF CHR ($(0), 1) = CHR ($(1),1) THEN GOTO  100
```

CLEAR

Statement (Direct/Program)

FUNCTION — Initializes variables, interrupts, and stack information.

DESCRIPTION — Executing the CLEAR statement initializes all variables. A '0' is substituted in the variables. The interrupt of the ONTIME statement is also initialized.

The real-time clock of the ASCII module started by the CLOCK1 statement is not affected even if the CLEAR statement is executed. The area for character string variables allocated by the STRING statement is not affected either.

Example:

```
>LIST
10    PRINT "*** ASCII module REAL TIME CLOCK ***"
20    STRING 100,20
30   $(0)="ASCII/BASIC module " : $(1)='ASC-6210"
40   TIME=0 :  ONTIME 2,40
50    CLOCK 1
60    PRINT TIME,"SECONDS" :  PRINT $(0),$(1)
70   VAR=INT(RND*100) :  PRINT "VAR= ",VAR
80    END

READY
>RUN

*** ASCII module REAL TIME CLOCK ***
.605 SECONDS
ASCII/BASIC module  ASC-6210
VAR=  36

READY
>CLEAR

>PRINT TIME,"SECONDS":PRINT "VAR= ",VAR
 30.835 SECONDS
VAR=  0
```

◄ The real-time clock is not affected after the CLEAR statement is evecuted. A variable is initialized to '0'.

```
>PRINT "$(0)=",$(0):$(0)="ASCII module":PRINT "$(0)=",$(0)
$(0)=
$(0)=ASCII module
>
```

◄ The area for character string variables is not affected after the CLEAR statement is executed.

NOTE

Do not use the CLEAR statement when the watchdog timer is operated with the ASCII module by operating the IE register. This may cause misoperation of the ASCII module.

CLEAR I

Statement (Direct/Program)

FUNCTION — Initializes all interrupts started by the BASIC-52 program.

DESCRIPTION — Executing the CLEAR I statement initializes interrupts of the ONTIME statement. The real-time clock of the ASCII module started by the CLOCK1 statement is not affected even if the CLEARI statement is executed.

Example:

```
>LIST
10      REM *** CLEAR INTERRUPTS ***
20      TIME=0
30      CLOCK 1
40      ONTIME 1, 100
50      DO
60      IF TIME>5 THEN CLEAR I
70      WHILE TIME<10
80      END

110     ONTIME TIME+1, 100
120     RETI


READY
>RUN

BASIC-52 TIMER INTERRUPTION AT  1 SEC.
BASIC-52 TIMER INTERRUPTION AT  2 SEC.
BASIC-52 TIMER INTERRUPTION AT  3 SEC.
BASIC-52 TIMER INTERRUPTION AT  4 SEC.
BASIC-52 TIMER INTERRUPTION AT  5 SEC.

READY
>
```

Do not use the CLEAR I statement when the watchdog timer is operated with the ASCII module by operating the IE register. This may cause misoperation of the ASCII module.

CLEAR S

Statement (Direct/Program)

FUNCTION — Initializes all stacks used by BASIC-52.

DESCRIPTION — Initializes the values of the three stacks used by BASIC-52, control (C-), argument (A-), and internal (I-) stacks.

Used to clear stacks if an error occurs during a subroutine. Provides an exit from a loop of a FOR ~ NEXT, DO ~ UNTIL, or DO ~ WHILE statement.

Example:

```
>LIST
10        REM *** CLEAR STACKS ***
20        FOR I= 1 TO 10
30        PUSH I
40        IF I=7 THEN CLEAR S : GOTO 100
50        NEXT I
100       PRINT "I=", I
110       POP I1
120       PRINT I1
130       END

READY
>RUN

I= 7

ERROR: A-STACK - IN LINE 110

110          POP I1
-------------X
READY
>
```

◄ The A-STACK error occurred because popping data was attempted although the argument stack was initialized in line 40.

CLOCK 1

Statement (Direct/Program)

FUNCTION — Switches on the real-time clock.

DESCRIPTION — Executing this statement switches on the real-time clock. The real-time clock value can be read and set using the TIME function. The TIME function value increases every 5 msec after executing this statement. The TIME function value can be between 0 and 65535.995 sec. Overflow occurs, and the count returns to 0, if the TIME function value exceeds 65535.995.

Do not use the CLOCK1 statement when the watchdog timer is operated with the ASCII module by operating the IE register. This may cause misoperation of the ASCII module.

CLOCK 0

Statement (Direct/Program)

FUNCTION — Switches off the real-time clock.

DESCRIPTION — Executing this statement switches off the real-time clock. The value of the TIME function does not increase after executing this statement.

The real-time clock is not switched off even if the CLEAR or CLEAR I statement is executed.
Do not use the CLOCK0 statement when the watchdog timer is operated with the ASCII module by operating the IE register. This may cause misoperation of the ASCII module.

CONT

Command

FUNCTION

Resumes execution of a program whose execution has been interrupted by Control-C or by the STOP statement.

DESCRIPTION

This statement resumes the execution of a program whose processing has been interrupted by the input of Control-C or by the execution of the STOP statement.

If execution has been interrupted with the prompt output by an INPUT statement, the prompt ('?' or a prompt character string) is redisplayed and program execution is resumed.

Execution can be resumed after changing numerical variable or character string variable data during program execution interrupt. Execution cannot be resumed if the program itself is changed during execution stop. A CAN'T CONTINUE error will result.

Example:

```
>LIST
10        INPUT "ENTER 2 VARIABLES " ,A,B
20        A1=A**2: B1=B**3
30        PRINT "A**2=" ,A1
40        PRINT "B**3=" ,B1
50        GOTO 10

READY
>RUN

ENTER 2 VARIABLES 20, 7 [CR]
A**2=400
B**3=343
ENTER 2 VARIABLES Control-C
STOP-IN LINE 10
READY
>CONT

ENTER 2 VARIABLES 26, 9 [CR]
A**2=676
B**3=729
ENTER 2 VARIABLES Control-C
STOP - IN LINE 10
READY
>
```

COS (<Mathematical Expression>)

Function

FUNCTION — Provides cosine (cosine function) values. (y = cos x)

DESCRIPTION — Provides cosine value of angle given by <mathematical expression>. The angle unit is radian. Calculations are performed in 7 digits of valid numerals. Set angles are expressed by <mathematical expression> in the range of ±200000 radian.

CR

Function

FUNCTION — Moves the cursor to the top of ther current line.

DESCRIPTION — If a CR function is used during PRINT statement execution, a carriage return (CR) is performed, but line feed (LF) is not performed during output.

For example, data that is repetitively updated can be displayed on the CRT in one line.

Example:

```
>10     PRINT SPC(1), "I"
>20     FOR I=1 TO 100
>30     PRINT I, CR,
>40     NEXT I
>RUN
```

I
□ The loop counter I value is displayed in □ after it is continually updated.

DATA <Numerical Constant> ~ READ <Variable> ~ RESTORE

Statement (Program)

FUNCTION

Reads numerical values set by the DATA statement in variables set by the READ statement. The RESTORE statement returns the pointer that indicates the data in the DATA statement to be read next by the READ statement to the top of the DATA statement.

DESCRIPTION

A DATA statement sets numerical constants read by a READ statement. Numerical constants that can be expressed by the form of <mathematical expression> are valid. Data that can be put in one line can be written in one DATA statement. Use commas (,) to delimit data.

The READ statement reads numerical constants set in the DATA statement from the beginning sequentially and allocates them to variables on a 1 to 1 basis. Variables that can be put in one line can be written in one READ statement. Use commas (,) to delimit variables. A NO DATA error results if the number of variables is larger than that of data in a DATA statement.

The RESTORE statement returns the internal pointer that shows the data in the DATA statement to be read next in variables by the READ statement to the top of the DATA statement.

Example:

```
>LIST
10        REM DATA-READ-RESTORE
20        FOR I=1 TO 3
30        READ A,B :  PRINT A,B
40        NEXT I
50        READ A,B :  PRINT A,B
60        RESTORE
70        READ A,B :  PRINT A,B
80

READY
>RUN

10  20
 3.1415926  2.7182818
.5  4
4.5777066 E-5  1
10  20

READY
>
```

DBY (<Mathematical Expression>)

Function

FUNCTION

Reads and sets data of the internal data memory of the ASCII module.

DESCRIPTION

Reads data of the internal data memory of the ASCII module and sets values.

Values between 0 and 255 (0FFH) are valid as the range of values expressed by <mathematical expression>. Values between 0 and 255 (0FFH) are valid as set values also. A BAD ARGUMENT error will occur if values are outside this range.

WARNING

The ASCII module may operate incorrectly if the value of any address of the internal data memory is set with a DBY( ) function. Do not set values other than in area shown by "ASCII Module Memory Layout" in APPENDIX 6. INFORMATION ABOUT THE ASCII MODULE.

Example:

```
>PRINT DBY(15H)
 0
>DBY(15H) = 100

>
```

DIM

Statement (Direct/Program)

FUNCTION

Designates the size of elements of array variables and allocates memory regions.

DESCRIPTION

A memory area for array variables is secured for the designated size. A '0' is set for array variables as an initial value. Only linear arrays can be used with BASIC-52, and the maximum size of subscripts is 254. The subscripts start with 0, so the actual number of elements in each array will be (number of subscripts + 1).

If an array variable is used without performing array declaration by a DIM statement, an array making the size of subscripts to 10 will be allocated automatically.

If the value of subscripts surpasses the preset maximum value, an ARRAY SIZE error will result. The array variables declared by a DIM statement can be deleted by the RUN command, NEW command, or CLEAR statement and returned to an initial state.

Example:

```
>LIST
10        DIM A(15)
20        B(5)=50
30        FOR I=0 TO 15
40        A(I)=I : PRINT A(I),
50        NEXT I
60        PRINT
70        FOR J=0 TO 15
80        PRINT B(J),
90        NEXT J

READY
>RUN

0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
0  0  0  0  0  50  0  0  0  0  0

ERROR:  ARRAY SIZE - IN LINE  80◄ The size of the array variable B is
                                  assigned default of 10.
80        PRINT B(J),             The ARRAY SIZE error occurred
----------  X                     because accessing the over-sized
READY                             array variable was attempted.
>
```

DO ~ UNTIL <Relational Expression>

Statement (Program)

FUNCTION

Statements contained from the DO through the UNTIL statements are repetitively executed until the condition given in the UNTIL statement is met.

DESCRIPTION

Statements sandwiched between the DO and UNTIL statements are repetitively executed until the relational expression given in the UNTIL statement becomes true. The DO-UNTIL loop can be nested.

DO ~ WHILE <Relational Expression>

Statement (Program)

FUNCTION

Statements contained between the DO and WHILE statements are repetitively executed as long as the condition given in the WHILE statement is true.

DESCRIPTION

Statements sandwiched between the DO and WHILE statements are repetitively executed as long as the relational expression given by the WHILE statement is true. The DO-WHILE loop can be nested.

Example:

```
>LIST

10      DO
20      I=I+1
30      DO
40      J=J+1
50      PRINT " I=",I," J=",J," I+J=",I+J
60      WHILE J<3
70      UNTIL I=3
80      END

READY
>RUN

I= 1    J= 1    I+J= 2
I= 1    J= 2    I+J= 3
I= 1    J= 3    I+J= 4
I= 2    J= 4    I+J= 6
I= 3    J= 5    I+J= 8

READY
>
```

END

Statement (Program)

FUNCTION — Ends program execution and returns to the direct mode.

DESCRIPTION — Ends program execution and returns to the direct mode.

If a program does not have an END statement, BASIC-52 automatically ends the program after executing the statement on the last line of the program.

Example:

```
10    FOR I=0 TO 3
20    PRINT I*2
30    NEXT I
40    PRINT "I=", I

RUN
0
2
4
6
I=4

READY
>
```

```
10    FOR I=0 TO 3
20    PRINT I*2
30    NEXT I
35    END
40    PRINT "I=", I

RUN
0
2
4
6

READY
>
```

EXP (<Mathematical Expression>)

Function

FUNCTION

Provides value of exponential function to 'e' (base of natural logarithms). ($y = e^x$)

DESCRIPTION

Returns result of exponential function when value expressed by <mathematical expressions> is used as exponent and base is used as e (= 2.7182818).

Values from -255 to 255 are valid as ranges of values expressed by <mathematical expressions>. If the value is outside this range, a BAD ARGUMENT error will result.

FOR ~ TO ~ [STEP] ~ NEXT

Statement (Direct/Program)

FUNCTION

Statements contained between the FOR and NEXT statements are repetitively executed until the variable given in the FOR statement reaches the set value.

DESCRIPTION

The FOR ~ TO [STEP] ~ NEXT statements are used in the following format:

```
10    FOR A=B TO C STEP D
20    PRINT A,
30    NEXT A
40    END
```

A to D are allocated as follows:

A: Counter of FOR ~ TO [STEP] ~ NEXT loop
B: Loop counter initial value
C: Loop counter conclusive value
D: Loop increment

First, initial value B is set in the loop counter A. When the program execution of BASIC-52 proceeds to the NEXT statement, the value of increment D is added to the loop counter. The loop counter value and conclusive value C are compared. If the loop counter value is less than the conclusive value, the execution returns to the statement following the FOR statement.

If the loop counter value reaches or surpasses the conclusive value, the execution proceeds to the statement following to the NEXT statement.

The STEP statement and increment D can be omitted. In this case, the increment is automatically increased by 1. A negative value can be designated with the increment D. (Example: FOR I = 100 TO 10 STEP -2.)

Example:

```
10  FOR I= 1 TO 5
20  PRINT I,
30  NEXT I
40  PRINT
50  PRINT "I= ", I

RUN
1  2  3  4  5
I= 6
```

```
10  FOR I= 10 TO 2 STEP -3
20  PRINT I,
30  NEXT I
40  PRINT
50  PRINT "I= ", I

RUN
10  7  4
I= 1
```

The decision of the FOR ~ TO ~ [STEP] ~ NEXT statements is made at the end of the loop. In the following cases, the statements between the FOR and NEXT statements are executed only once, and the execution proceeds to the statement following the NEXT statement.

1. The increment value is positive and the initial value is larger than the conclusive value.

```
FOR I = 10 TO 1 STEP 2
```

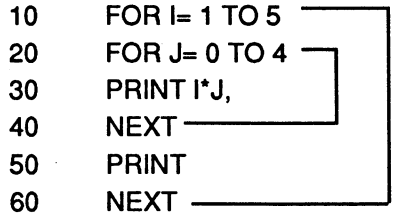
2. The increment value is negative and the initial value is smaller than the conclusive value.

```
FOR J = 1 TO 5 STEP -1
```

Looping extends indefinitely unless the loop counter is set to a value larger than the conclusive value inside the loop if the increment value iw is 0.

```
FOR K = 1 TO 5 STEP 0
```

The loop counter in the NEXT statement can be omitted. In this case, a pair is formed with the nearest FOR statement.



The FOR ~ TO ~ [STEP] ~ NEXT statements can be nested. In BASIC-52, the maximum number of nesting operations is 9. A C-STACK error results if nesting is performed in excess of this limit or if FOR and NEXT statements do not match correctly.

FREE

System Variable

FUNCTION — Shows how many bytes are unused in the RAM.

DESCRIPTION — Shows how many bytes are unused in the RAM. Only value reading is possible; value setting is not possible. If a program does not exist in the RAM, the relationship of

FREE = MTOP - 512

will establish. If a program exists in the RAM, the relationship of

FREE = MTOP - LEN - 511

will establish if the BASIC-52 interpreter is in the RAM mode.

Example:

```
>RAM

READY
>NEW

>PRINT MTOP,:PRINT LEN,:PRINT FREE
29695 1 29183

>10 FOR I=1 TO 10
>20 PRINT I,
>30 NEXT I
>40 END
>

>PRINT MTOP,:PRINT LEN,:PRINT FREE
29695 30 29154

>
```

GET

Function

FUNCTION

Reads input by the equipment connected to CH1 of the ASCII module and gives the character codes of the input characters if the input is valid.

DESCRIPTION

The GET function has a meaning only in the program mode. A '0' is always returned if the GET function value is read in the direct mode. If the GET value is read during program execution, the GET value becomes 0 unless the equipment performs input again.

Example:

```
>LIST
10        A=GET
20        IF A=0 THEN GOTO 10
30        PRINT CHR (A), A, :  PH0. A
40        GOTO 10

READY
>RUN

A         65        41H
B         66        42H
1         49        31H
9         57        39H
Control-C
STOP-IN LINE 10
READY
>
```

This message is displayed each time "A", "B", "1", or "9" is input by the console device keyboard.

GOSUB ~ RETURN

Statement (Program)

FUNCTION

Calls a subroutine beginning with the line No. after GOSUB. During subroutine execution, the execution returns to the statement following the GOSUB statement in the main program when the RETURN statement is executed.

DESCRIPTION

The RETURN statement returns the execution to the statement following the GOSUB statement that called the subroutine. Therefore, several RETURN statements can be contained in the subroutine program.

Subroutine nesting by calling other subroutines during one subroutine execution is also possible. The maximum number of nesting operations with BASIC-52 is 52. A C-STACK error results if the RETURN statement is executed before the GOSUB statement is executed or if nesting is performed in excess of this limit.

Example:

```
>LIST
10      FOR I=0 TO 3
20      GOSUB 100
30      NEXT I
40      END
100     PRINT I,
110     GOSUB 200
120     RETURN
200     PRINT EXP (I)
210     RETURN

READY
>RUN


0       1
1       2.7182818
2       7.3890559
3       20.085536

READY
>
```

```
10      FOR I=0 TO 3
20      GOSUB 100
30      NEXT I
100     PRINT I,
110     GOSUB 200
120     RETURN
200     PRINT EXP (I)
210     RETURN

READY
>RUN

0       1
1       2.7182818
2       7.3890559
3       20.085536
4       54.598146

ERROR: C-STACK-IN LINE

120   RETURN
--------------X
READY
>
```

The following program forcibly escapes from the loop of the FOR ~ TO ~ [STEP] NEXT statements during subroutine execution.

```
READY
>LIST
10          INPUT "VAR?", X
20          FOR I=1 TO 10
30          GOSUB 100
40          NEXT I
50          PRINT "END!"
60          END
100         FOR J=0 TO I
110         IF X>100 THEN 140
120         PRINT X*J,
130         NEXT J: PRINT
140         RETURN

READY
>RUN

VAR? 7
0     7
0     7     14
0     7     14    21
0     7     14    21    28
0     7     14    21    28    35
0     7     14    21    28    35    42
0     7     14    21    28    35    42    49
0     7     14    21    28    35    42    49    56
0     7     14    21    28    35    42    49    56    63
0     7     14    21    28    35    42    49    56    63    70
END!

READY
>RUN

VAR? 207
END!

READY
>
```

GOTO ~

Statement (Direct/Program)

FUNCTION

Unconditionally transfers program execution control to the line No. that follows GOTO.

DESCRIPTION

Transfers execution to the line designated after GOTO. If the designated line is a non-executable statement, such as a REM or DATA statement, execution starts beginning with the first executable statement thereafter.

Executing a GOTO statement in the direct mode allows program execution to be started at any point. In this case, variables are not initialized and interrupts are not cleared. By resuming program execution by a GOTO statement after editing a BASIC-52 program, variables are initialized and interrupts are cleared.

An INVALID LINE NUMBER error will result if the line No. designated by the GOTO statement does not exist in the program.

Example:

```
READY
>LIST
10      I=0
20      GOTO 100
100     PRINT "I=", I
110     I=I+1
120     GOTO 200
200     PRINT "I=", I
210     I=I+1
220     GOTO 300
300     PRINT "I=", I

READY
>RUN

I=0
I=1
I=2

READY
>
```

```
>LIST
10      I=0
20      GOTO 100
100     PRINT "I=", I
110     I=I+1
120     GOTO 200
200     PRINT "I=", I
210     I=I+1
220     GOTO 300

READY
>RUN

I=0
I=1


ERROR: INVALID LINE NUMBER-IN LINE 220

220     GOTO 300
----------------X
READY
>
```

IE

Special Function

FUNCTION

Reads and sets value of the 8025AH IE register and activates the watchdog timer of the ASCII module.

DESCRIPTION

The IE register is an interrupt enable register. As one of the RAS functions, the ASCII module has the watchdog timer function. This IE register sets and releases the watchdog timer function.

The watchdog timer operation is:

IE = IE .OR.2

Of the BASIC-52 commands, the values of the CLOCK0, CLOCK1, CLEAR and CLEARI commands are also set in the IE register. Do not use these commands if the watchdog timer is activated.

IF ~ THEN ~ ELSE

Statement (Program)

FUNCTION

Decides conditions of the relational expression that follows the IF statement and changes the program flow in accordance with the result of the decision.

DESCRIPTION

The IF ~ THEN ~ ELSE statements are used in the following format:

```
10 IF A<0  THEN A=0  ELSE GOTO 100
```

Allocation is made as follows:

between IF and THEN:	relational expression
between THEN and ELSE:	BASIC-52 statement array or line No.
after ELSE statement:	BASIC-52 statement array or line No.

A format of 10 IF A=100 GOTO 100 ELSE GOTO 200 is also possible. The ELSE part of the expression can be omitted also. The relational expression that follows IF is evaluated. If the expression is true, parts after THEN are executed. If the expression is false, parts after THEN or GOTO are ignored and the part after ELSE is executed, if it exists.

Example:

```
10      INPUT "INITIAL?", A
20      IF A<0 THEN A=ABS(A) ELSE A=A*2
30      IF A=0 THEN GOTO 100 ELSE GOTO 200
 •
 •
 •
100     FOR I=A TO 10
 •
 •
 •
200     PRINT "A=", A
 •
 •
```

(a)

```
10      INPUT "INITIAL?", A
20      IF A<0  A=ABS(A) ELSE A=A*2
30      IF A=0 THEN 100 ELSE 200
  •
  •
  •
100     FOR I=A TO 10
  •
  •
  •
200     PRINT "A=", A
  •
  •
  •
```

(b)

(a) and (b) perform exactly the same processing

INPUT

Statement (Program)

FUNCTION

Reads numerical values, character strings, and other items from the equipment connected to CH1 of the ASCII module and substitutes them in variables.

DESCRIPTION

This statement reads data from the console device connected to CH1 of the ASCII module during the execution of a BASIC-52 program. It awaits input from the console if the INPUT statement is executed. Input from the console ends by pressing the [CR] key, and execution then proceeds to the next statement.

Example:

```
10      INPUT    A
20      PRINT    A

RUN

?10     [CR]
 10

READY
>
```

Data can be set in two or more variables in one INPUT statement. In this case, the variables that follow the INPUT statement must be delimited by commas (,). Data input by the console must also be delimited by commas (,). The message 'TRY AGAIN' will be displayed, and the system will again await for input if the number or format of the data input by the console does not match the number or format of variables in the INPUT statement.

```
10      INPUT    A, B
20      PRINT    A, B

RUN

?1      [CR]

TRY AGAIN

?1, A   [CR]

TRY AGAIN

?1,     10 [CR]
 1      10
READY
```

A PROMPT statement can be inserted between an INPUT statement and variables to make the type of the data to be input more understandable. If a comma (,) is put after the PROMPT statement, the question mark will not be displayed.

```
10  INPUT "INPUT A VARIABLE " A
20  PRINT A*2

RUNRUN

INPUT A VARIABLE
? 10 [CR]
  20

READY
>
```

```
10  INPUT "INPUT A VARIABLE ", A
20  PRINT A*2


INPUT A VARIABLE  10 [CR]
 20

READY
>
```

Character string variables can be read using an INPUT statement.

```
READY
>LIST
10  STRING 100, 20
20  $(0)="Hello,"
30  $(2)="My name is ASC-6210."
40  INPUT "May I have your name, please?", $(1)
50  PRINT $(0), $(1)
60  PRINT $(2)

READY
>RUN
May I have your name, please? EX2000
Hello, EX2000
My name is ASC-6210.

READY
>
```

If there is only one variable, a null string is substituted in the character string variable if only the [CR] key is input.

For a numerical variable, input of only the [CR] key is regarded unsuitable and the message 'TRY AGAIN' will be displayed.

INS <Integer>

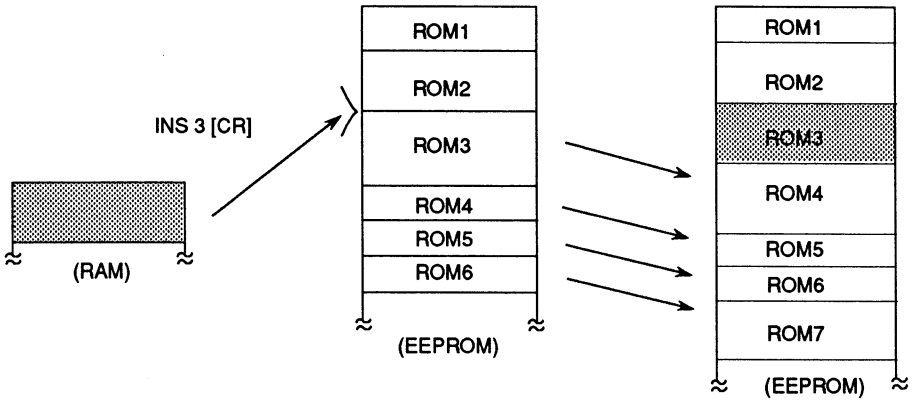
Command

FUNCTION — Inserts a BASIC-52 program stored in the RAM into the designated position program No. of the EEPROM.

DESCRIPTION — Inserts the BASIC-52 program in the RAM into the position of the program No. designated after INS. The Nos. of the programs after the program inserted are automatically shifted backward.

Example:

```
>LIST
10        STRING 100,20
20       $(0)="Hello,"
30       $(2)="My name is ASC-6210."
40        INPUT "May I have your name, please?",$(1)
50        PRINT $(0),$(1)
60        PRINT $(2)
70        PRINT : GOTO 40

READY
>INS 3
INSERT ROM 3 OK?(ENTER) [CR]
JUST A MOMENT, PLEASE...
COMPLETED.

READY
>
```



INS processing is not performed if key input other than the [CR] key is made when the message "INSERT ROM n OK? (ENTER)" is displayed.

The error messages regarding the INS command are:

- **BAD SYNTAX**

  The INS command was used in the program mode.

- **CAN'T SAVE PROGRAMS ANY MORE.**

  There are already 255 programs saved, and no additional programs can be inserted.

- **EEPROM MEMORY**

  An error occurred during writing of BASIC-52 programs in the EEPROM.

- **INVALID ROM NO.**

  The ROM program No. designated by the INS command was invalid.

- **NO PROGRAMS ON RAM.**

  Insertion was attempted although there were no programs in the RAM.

- **PROGRAM > REST OF EEPROM FREE AREA.**

  The program in the RAM was longer than the remaining area in the EEPROM and so was not inserted.

INT (<Mathematical Expression>)

Function

FUNCTION

Provides the integer part of a <mathematical expression> value.

Example:

```
100       PRINT "   X          INT( )"
110       FOR I=-9.80665 TO 8.31441 STEP 2.5
120       PRINT I, SPC (2), INT (I)
130       NEXT I
140       END

READY
>RUN

     X      INT( )
-9.80665   -9
-7.30665   -7
-4.80665   -4
-2.30665   -2
.19335     0
2.69335    2
5.19335    5
7.69335    7

READY
>
```

KILL <Integer>

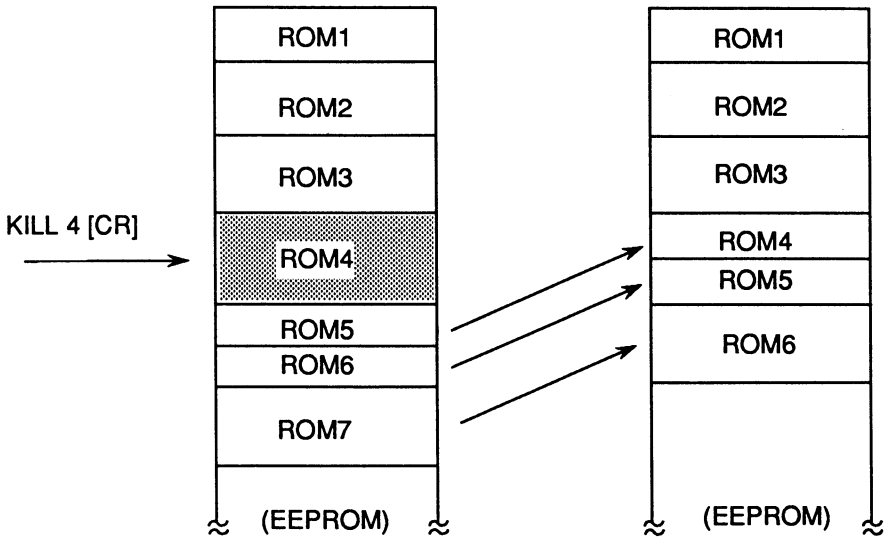
Command

FUNCTION — Deletes the BASIC-52 program saved in the designated EEPROM position (program No.).

DESCRIPTION — Deletes the BASIC-52 program saved in the position of the program No.designated after KILL. The Nos. of the programs after the programs deleted are automatically shifted forward.

Example:

```
>KILL 4
KILL ROM 4 OK?(ENTER) [CR]
JUST A MOMENT, PLEASE...
COMPLETED.

READY
>
```



Killing is not performed if key input other than the [CR] key is made when the message "KILL ROM n OK? (ENTER)" is displayed.

The error messages regarding the KILL command are:

- **BAD SYNTAX**

  The KILL command was used in the program mode.

- **EEPROM MEMORY**

  An error occurred during deletion of BASIC-52 programs from EEPROM.

- **INVALID ROM NO.**

  The ROM program No. designated by the KILL command was invalid.

LD@ <Mathematical Expression>

Statement

FUNCTION

Pushes values stored at address designated by the mathematical expression that follows LD@ to the argument stack.

DESCRIPTION

The memory addresses expressed by the <mathematical expression> must be ones that are effective inside the ASCII module and that do not destroy work areas of the ASCII module and of BASIC-52. Effective addresses are addresses 205H to MTOP (normally address 73FFH).

By combining this command and the ST@ command, data pushed to the argument-stack can be transferred.to other memory areas.

Example:

```
READY
>LIST
10        REM *** STORE DATA ***
20        FOR I=1 TO 3
30        INPUT "DATA TO BE STORED - ",A
40        PUSH A : ST@ 7350H+6*I
50        NEXT I
60        PRINT
70        REM *** LOAD DATA ***
80        FOR I=1 TO 3
90        LD@ 7350H+6*I : POP A (I)
100       PRINT "STORED DATA = ",A(I)
110       NEXT I

READY
>RUN

DATA TO BE STORED - 100
DATA TO BE STORED - 3.1415926
DATA TO BE STORED - -207

STORED DATA = 100
STORED DATA = 3.1415926
STORED DATA = -207
READY
>
```

The memory addresses shown by ST@ and LD@ are the highest bytes of areas to store values from the A-stack. (An area of 6 bytes is needed for one value). In the above-mentioned example, the first data will be ST@ (7350H+6) and will be stored at 7356H, 7355H, 7354H, 7353H, 7352H and 7351H of the external RAM.

LEN

System Variable

FUNCTION

Provides the length of the BASIC-52 program currently selected.

DESCRIPTION

Provides the length of the program created and edited in the RAM if the BASIC-52 is in the RAM mode. The length of the program selected is given if a program in the EEPROM is selected by the PROM command. Only reading of values is possible. Setting of values is not possible. A '1' will be returned as the LEN value if a program does not exist.

LET

Statement (Direct/Program)

FUNCTION

Substitutes values given by <mathematical expression> in variables.

DESCRIPTION

For example, in the statement

LET FX=X* (1-X/2+X**2/3-X**3/4),

the result of the mathematical expression X* (1-X/2 +X**2/3-X**3/4) is substituted in variable FX. LET can be omitted. Even if the foregoing statement is changed to

FX=X*(1-X/2+X**2/3-X**3/4),

the BASIC-52 interpreter performs the same processing.

Example:

```
10  STRING 50, 10                10  STRING 50, 10
20  LET A=SIN (PI)+COS(PI)       20  A=SIN(PI)+COS(PI)
30  LET $(0)="INPUT DATA"        30  $(0)="INPUT DATA"
 :                                :
 .                                .
```

(a) LET statement is used. (b) LET statement is omitted.

(a) and (b) perform exactly the same operation.

LIST [Line No.] [-Line No.]

Command

FUNCTION — Outputs the current program list on the equipment connected to CH1 of the ASCII module.

DESCRIPTION — Outputs the current program list on the equipment connected to CH1 of the ASCII module. List output can be stopped by inputting Control-C during list output.

Example:

```
>LIST
10          PRINT "DO-UNTIL LOOP"
20          I=0
30          DO
40          PRINT I,
50          I=I+1
60          UNTIL I=10
70          END

READY
>
```

The LIST command can also be used in the following two ways.

List <Start line No.>

List <Start line No.> - (dash) < End line No.>

Example:

```
>LIST 30
30          DO
40          PRINT I,
50          I=I+1
60          UNTIL I=10
70          END

READY
>LIST 30-60
30          DO
40          PRINT I,
50          I=I+1
60          UNTIL I=10

READY
>
```

LIST #

Command

FUNCTION — Outputs the current program list to the equipment connected to CH2 of the ASCII module.

DESCRIPTION — Outputs the current program list to the LIST device connected to CH2 of the ASCII module. Be sure to set a baud rate for CH2 by the BAUD statement before executing the LIST# command. The cautions and usage variations mentioned in the description of the LIST command apply directly to the LIST# command.

Example:

```
10      PRINT "DO-WHILE LOOP"
20      I=0
30      DO
40      PRINT I,
50      I=I+1
60      WHILE I<11
70      END

READY
>BAUD 300

READY
>LIST #

(Program list is output on equipment connected to CH2.)

READY
>
```

LOG (<Mathematical Expression>)

Function

FUNCTION — Provides value of natural logarithm. (y = loge , y = ln x)

DESCRIPTION — Returns natural logarithm (logarithms using e as base) of value expressed by <mathematical expression> as function value.
Values between 0 and EXP (255) (=5.5602087 E + 110 are valid as the range of values expressed by <mathematical expression>. A BAD ARGUMENT error will result if the value is outside of this range.

Example:

```
>LIST
10        PRINT USING(##.######)
20        PRINT" X EXP(X)  LOG(X) EXP(LOG(X))  LOG(EXP(X))"
30        E=2.7182818
40        FOR I=1 TO 2
50        PRINT I,EXP(I),LOG(I),EXP(LOG(I)),LOG(EXP(I))
60        NEXT I
70        PRINT E,EXP(E),LOG(E),EXP(LOG(E)),LOG(EXP(E))

READY
>RUN

     X    EXP(X)    LOG(X)       EXP(LOG(X))  LOG(EXP(X))
1.000000 2.718281  0.000000      1.000000      1.000000
2.000000 7.389055  0.693147      2.000000      2.000000
2.718281 15.154261 1.000000      2.718281      2.718281

READY
>
```

MTOP

System Variable

FUNCTION

Reads and sets the value of the last address of the RAM area allocated for BASIC-52 programs.

DESCRIPTION

BASIC-52 programs and data such as variables used in programs are created and set within the range not exceeding MTOP. The length of BASIC-52 programs and data storage areas is given by the following equation:

(BASIC-52 program data area) = MTOP - 512 (bytes)

CAUTION A value can be set such as MTOP = 2000. Note that setting a value without carefully studying the ASCII module memory layout may cause misoperation of the ASCII module.

NEW

Command

FUNCTION

Deletes a program in the RAM and clears all the variables.

DESCRIPTION

The NEW command deletes a program in the RAM. It clears all numerical variables to 0, clears character string variables (strings) to null strings, and clears all interrupts.

It does not clear the real-time clock, setting of character string variable storage areas, or internal stack pointer values.

Example:

```
>LIST
10              STRING 100,20
20             $(0)="Hello, "
30             $(2)="My name is ASC-6210."
40             INPUT "May I have your name, please? ",$(1)
50             PRINT $(0),$(1)
60             PRINT $(2)
70             PRINT : GOTO 40

READY
>RUN

May I have your name, please?  EX2000  [CR]
Hello, EX2000
My name is ASC-6210.

May I have your name, please?  Control-C
STOP - IN LINE 40
READY
>NEW

>LIST

READY
>$(0)="ASCII/BASIC module ":PRINT $(0)
ASCII/BASIC module ◄ The area for character string variables is not
                     affected after the NEW command is executed.
>
```

NOT (<Mathematical Expression>)

Function

FUNCTION — Provides 1's complement of a <mathematical expression> value as a function value.

DESCRIPTION — Values expressed by <mathematical expression> must be integers between 0 and 65535 (0FFFFH). A BAD ARGUMENT error will result if the value is outside the range 0 to 65535. If the numerical value is not an integer, numerals following the decimal point are truncated.

Example:

```
>PRINT NOT(0), NOT(1), NOT(3.14), NOT(65535)
65535  65534  65532  0
```

NULL <Integer>

Command

FUNCTION

Sets the number of NULL characters (100H) that are to be output after carriage return.

DESCRIPTION

Outputs the number of NULL characters set by the integer that follows NULL after a carriage return. Integer values from 0 to 255 are valid. NULL = 0 is set when power is turned on. The number of NULL characters set by the NULL command is stored at Address 15H (21) in the internal RAM. Thus, the number of NULL characters can also be set by
DBY (15H) = <mathematical expression>
without using the NULL command.

ON <Mathematical Expression> GOSUB ~

Statement (Program)

FUNCTION — Calls the subroutine beginning with the line No. that follows GOSUB in accordance with the expression value.

DESCRIPTION — The expression value decides which line No. in the program to branch to.

Example:

In the statement
10 ON C GOSUB 100, 200, 300,
the program branches to line No. 100 if the value of C is 0, to line No. 200 if the value of C is 1, and to line No. 300 if the value of C is 2.

From the line Nos. 200, 300, and 400, the program must be subroutines that end with a RETURN statement.

ON <Mathematical Expression> GOTO ~

Statement (Program)

FUNCTION — Transfers the program execution control to the line No. that follows GOTO in accordance with the expression value.

Example:

In the statement
10 ON C1 GOTO 100, 200, 300,
the program branches to line No. 100 if the value of C1 is 0, to line No. 200 if the value of C1 is 1, and to line No. 300 if the value of C1 is 2.

The common errors occur in both the ON~GOSUB statement and ON~GOTO statement. They are as follows.

A BAD ARGUMENT error results if the value of the <mathematical expression> is negative.

A BAD SYNTAX error will occur if the number of branching operations given by the <mathematical expression> and the number of line Nos. do not match.

ONERR <Line No.>

Statement (Program)

FUNCTION

Enables interrupts that perform error processing and transfers the program execution control to the line No. that follows ONERR if an error occurs.

DESCRIPTION

If an error interrupt is enabled by this statement, processing designated by <line No.> is executed for errors that have occurred in the BASIC-52 program. The errors that can be handled by this statement are four types of mathematical errors: ARITH. OVERFLOW, ARITH. UNDERFLOW, DIVIDE BY ZERO, and BAD ARGUMENT.

If an error of any one of these four error types occurs during the program execution, the code of the error that has occurred is set at address 101H (257) of the external memory. The error codes are as:

10: DIVIDE BY ZERO error has occurred.
20: ARITH. OVERFLOW error has occurred.
30: ARITH. UNDERFLOW error has occurred.
40: BAD ARGUMENT error has occurred.

The data of these error codes can be read by the XBY( ) function.

Example:

```
>10     ONERR 100
>20     INPUT "DATA ?", A
>30     B=10
>40     PRINT B/A
>50     END
>100    PRINT "ERROR!", XBY(101H)
>110    END
>RUN

DATA?  0[CR]

ERROR! 10

READY
>
```

ONTIME <Mathematical Expression>, <Line No.> ~ RET I

Statement (Program)

FUNCTION

Declares that if the value of the real-time clock exceeds the value that follows ONTIME, an interrupt is performed and program execution control is transferred to the subroutine that starts with the line No. following the numerical value.

DESCRIPTION

An interrupt occurs if the special function TIME is equal to or larger than the value given by the <mathematical expression> that follows ONTIME. The comparison is made by picking up integer parts from both the TIME value and the value. The ONTIME statement uses the value of the TIME function.

Thus, the CLOCK 1 statement must be executed to switch on the real time clock. If the CLOCK 1 statement is not executed, the value of the TIME function does not increase and no changes occur even if the ONTIME statement is executed. (An error does not result.)

Be sure to return to the main program from the subroutine transferred by the ONTIME statement, by executing the RETI statement. Escape from the interrupt routine is not possible if the RETURN statement is used instead of the RETI statement.

Do not use the ONTIME statement when the watchdog timer is operated with the ASCII module by operating the IE register. This may causes misoperation of the BASIC-52 program.

PI

Constant

FUNCTION — Provides a circle ratio of (π) = 3.1415926.

Example:

```
>LIST
10        PRINT "PI =",PI
20        PRINT "RADIUS   SQUARE MEASURE"
30        FOR R=1 TO 5
40        S=PI*R**2
50        PRINT R, SPC (5),S
60        NEXT R
70        END

READY
>RUN

PI = 3.1415926

RADIUS   SQUARE MEASURE

  1.000000    3.1415926
  2.000000    12.56637
  3.000000    28.274333
  4.000000    50.265481
  5.000000    78.539815

READY
>
```

PH0. PH1.

Statement (Direct/Program)

FUNCTION

Displays values, variables, character strings, results of mathematical expressions, and other data on the equipment connected to CH1 of the ASCII module. Values between 0 and 65535 (0FFFFH) are displayed hexadecimally. PH1. always displays four hexadecimal digits. PH0. omits 0's in the upper two digits if the value is 255 (0FFH) or less.

DESCRIPTION

The suffix 'H' always follows numerals if a value between 0 and 65535 is displayed using the PH0. or PH1. statement, allowing you to distinguisth these statements from the PRINT statement. If the value is a real number, numerals following the decimal point are truncated. If a value is outside the range of 0 to 65535, the same format mode as the PRINT statement is automatically set.

The precautions and usage variations mentioned in the DESCRIPTION column of the PRINT and PRINT # statements also apply to the PH0. and PH1 statements.

Example:

```
>LIST
10    FOR I=1 TO 9 STEP 1.5
20    A=I**2:  B=I**3
30    PUSH I
40    PRINT I, TAB (10), A, TAB (20), :  PH0. A, TAB (30),
50    PRINT B, TAB (40), :  PH0. B, TAB (50), :  PH1. A, TAB (60), B
60    NEXT I

READY
>RUN

1     1       01H     1          01H     0001H     0001H
2.5   6.25    06H     15.625     0FH     0006H     000FH
4     16      10H     64         40H     0010H     0040H
5.5   30.25   1EH     166.375    A6H     001EH     00A6H
7     49      31H     343        157H    0031H     0157H
8.5   72.25   48H     614.125    266H    0048H     0266H

READY
>
```

PH0.#, PH1.#

Statement (Direct/Program)

FUNCTION

Outputs values, variables, character strings, results of mathematical expressions, and other data on the equipment connected to CH2 of the ASCII module. Values between 0 and 65535 (0FFFFH) are output hexadecimally. PH1.# always outputs four hexadecimal digits. PH0.# omits 0's in the upper two digits if the value is 255 (0FFH) or less.

DESCRIPTION

Results of mathematical expressions, character strings, and other data designated after the PH0.# and PH1.# statements are output on the equipment connected to CH2 of the ASCII module. The format for output is the same as that for the PH0. and PH1. statements. Be sure to set a baud rate for CH2 using the BAUD statement before executing PH0.# or PH1.# statements.

The precautions and usage variations mentioned in the DESCRIPTION column of the PRINT statement and PRINT# statement also apply to the PH0.# and PH1.# statements. Refer to the descriptions of those statements.

POP <Variable>

Statement (Direct/Program)

FUNCTION — Substitutes values popped from the argument stack in variables that follow POP.

DESCRIPTION — Values at top of the argument stack are substituted in variables that follow POP. Values are pushed to the argument stack by a PUSH statement or by the CALL routines.

Example:

```
10      REM *** EX WRITE ***
20      CALL 18:  POP A
30      PRINT "DATA  SENT  =", A
40      FOR I=1 TO A
50      CALL 2
60      POP B
70      PRINT B,
80      NEXT I
90      CALL 14
100     CALL 16
```

PORT1

Special Function

FUNCTION — Reads and sets values of the P1 I/O port of 8052AH.

DESCRIPTION — The second bit of PORT1 is an error bit. This bit can be used as an error bit in BASIC-52 programs if the ASCII module is used as an I/O module. Error bit is set and released as follows:

PORT1 = PORT1. OR. 0FFH:	Error bit set
PORT1 = 0FBH	Error bit release

PRINT/P./?

Statement (Direct/Program)

FUNCTION — Displays values, variables, character strings, results of mathematical expressions, and other data on the equipment connected to CH1 of the ASCII module. P. and ? (question mark) are short forms of the PRINT statement.

DESCRIPTION — Displays results of mathematical expressions, character strings, or other data designated after a PRINT statement on the equipment connected to CH1 of the ASCII module. A line feed is issued if nothing follows a PRINT statement.

Several pieces of data can be written after a PRINT statement by delimiting with commas (,). Data will be displayed with two spaces between each piece of data.

Example:

```
>PRINT PI, "ASCII/BASIC module", 10**3, 7.1E2
 3.1415926   ASCII/BASIC module   1000   710
```

PRINT#/P.#/?#

Statement (Direct/Program)

FUNCTION

Outputs values, variables, character strings, results of mathematical expressions and other data to the equipment connected to CH2 of the ASCII module. P.# and ?# are short forms of the PRINT# statement.

DESCRIPTION

Outputs results of mathematical expressions, character strings, and other data designated after the PRINT# statement to the equipment connected to CH2 of the ASCII module. Be sure to set a baud rate for CH2 using the BAUD statement before executing a PRINT# statement.

The precautions and usage variations mentioned in the DESCRIPTION column for the PRINT statement apply to the PRINT# statement also.

CAUTION

The execution of a PRINT# statement may be interrupted by a command sent by the EX CPU if a PRINT# statement is used by the ASCII module in interfacing with the EX CPU. Read the following to avoid this:

Do not use a PRINT# statement after executing CALL 20 during the READ mode.

Do not use the PRINT# statement if a command can be applied by the EX CPU (after executing CALL 13, 14 or 15).

PROM <Integer>

Command

FUNCTION

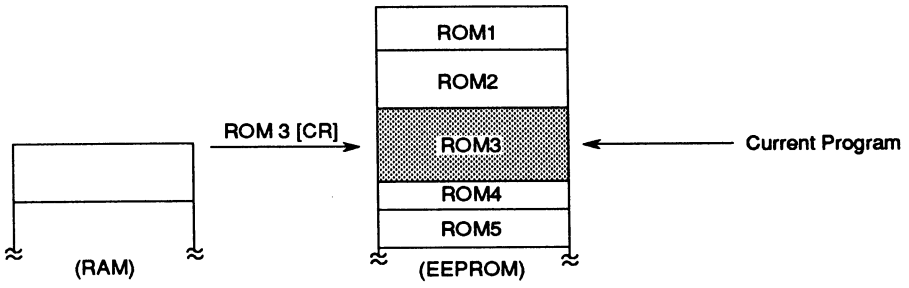
Selects the program of the designated program No. from the BASIC-52 programs saved in the EEPROM and sets BASIC-52 operation to the EEPROM mode.

DESCRIPTION

Selects the BASIC-52 program saved in the position designated after PROM as the current program and sets BASIC-52 operation to the EEPROM mode.

Example:

```
> PROM 3 [CR]

READY
> LIST [CR]

10          REM *** ROM 3 PROGRAM ***
20          INPUT "DATA SIZE ? " A
 .           .
 .           .
 .           .
```



BASIC-52 programs cannot be edited in the EEPROM mode. In order to edit the program in the EEPROM selected by the PROM command, first transfer that program to the RAM by the XFER command. Then editing is possible. The error messages related to the PROM command are:

- **INVALID ROM NO.**

  The ROM program No. designated during the PROM command was invalid.

- **MISSING OPERAND**

  The PROM command was used in the program mode.

PUSH <Mathematical Expression>

Statement (Direct/Program)

FUNCTION

Pushes mathematical expressions that follow PUSH to the argument stack (A-STACK).

DESCRIPTION

Pushes mathematical expressions that follow PUSH to the Argument Stack in succession.

PUSH statements hand over variables used during the CALL routines. They are also used to hand over values of variables to subroutines during a BASIC-52 program combined with a POP statement.

The value last pushed to the argument stack will be the value first fetched by a POP statement.

RAM

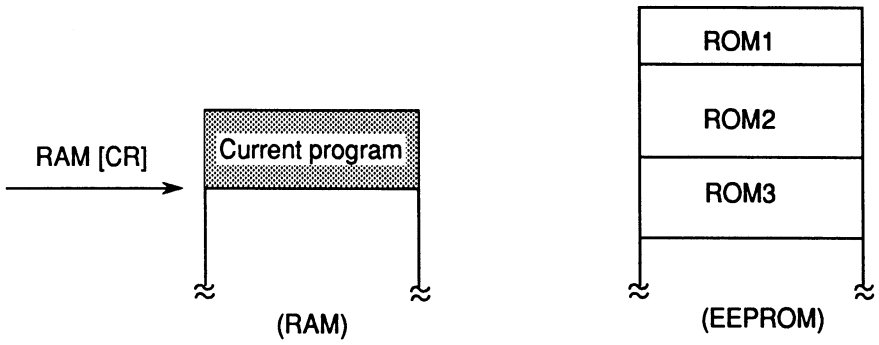
Command

FUNCTION — Sets BASIC-52 operation to the RAM mode.

DESCRIPTION — Sets BASIC-52 operation to the RAM mode. If a program exists in the RAM, that program is selected as the current program. If programs do not exist in the RAM, the RAM mode only is selected.

Example:

```
> LIST [CR]

10          REM *** ROM 3 PROGRAM ***
20          INPUT "DATA SIZE ?", A
 .           .
 .           .
 .           .

READY
> RAM [CR]

READY
> LIST [CR]

10          REM *** DATA DISPLAY PROGRAM ***
20          CALL 18
 .           .
 .           .
 .           .
```



RCAP2

Special Function

FUNCTION

Reads and sets values of registers RCAP2H and RCAP2L of 8052AH.

DESCRIPTION

The registers RCAP2H and RCAP2L are reloading registers for special function TIMER2. Pay attention when a value is set with function RCAP2. The RCAP2 value controls the baud rate of the serial port of CH1 of the ASCII module. The RCAP2 value and CH1 baud rate have the following relationship:
(CH1 baud rate) = XTAL/(32* (65536 - RCAP2))
The baud rate set based on the RCAP2 value is given priority even if a baud rate is set by the DIP switch on the front panel of the ASCII module if the RCAP2 value is changed afterward.

REM

Statement (Direct/Program)

FUNCTION

Add comments into a program.

DESCRIPTION

The REM statement is ignored during program execution. The REM statement is used to make a program easily understandable by adding comments into the program. The REM statement is valid as a jumping destination for GOTO, GOSUB and other statements. However, the execution is performed beginning the statement following the REM statement.

A REM statement can be used by delimiting the end of a line with a colon(:). However, if other statements are continued by delimiting parts after the REM statement with a colon(:), these other statements cannot be executed.

Example:

```
10    REM *** CALL 17 (CHAIN STATEMENT) ***
20    INPUT "ROM  No. ?", N
30    PUSH N
40    CALL 17:  REM EXECUTION →ROM (N) PROGRAM
50    END
```

RENUM [Integer] [,Integer]

Command

FUNCTION — Renumbers program line Nos. stored in the RAM.

DESCRIPTION — The RENUM command is used in the following format:

RENUM A, B

A: The starting line No. to be renumbered. If omitted, this will be 10.
B: Increment between line Nos. during renumbering. If omitted, this will be 10.

Example:

```
10      REM *** CALL 17 (CHAIN STATEMENT) ***
20      INPUT "ROM  NUMBER  ?  ", A
21      IF A < 0  GOTO 20
30      PUSH A
40      CALL 17
50      END

READY
> RENUM 100, 20

READY
> LIST

100     REM *** CALL 17 (CHAIN STATEMENT) ***
120     INPUT "ROM NUMBER  ?  ", A
140     IF A < 0 GOTO 120
160     PUSH A
180     CALL 17
200     END
```

An INVALID PARAMETER error results if parameters A and B used in the RENUM command are outside the effective range. The effective ranges for parameters A and B are:

A: 0 to 65535
B: 1 to 255

Line Nos. that can be used with BASIC-52 go up to 65535. A LIMIT OVER LINE NO. error results if a line No. larger than 65536 occurs during the execution of the RENUM command.
The RENUM command also changes line Nos. that are referred to in GOTO, GOSUB, THEN, ELSE and other statements. An INVALID LINE NO. IN XX error results if the referenced line No. is not written or does not conform to the BASIC-52 grammar. 'xx' is the line No. after RENUM. If the referenced line No. does not exist in the program, an UNDEFINED LINE NO. IN xx error results. 'xx' is the line No. after RENUM.

RND

Function

FUNCTION

Provides random numbers between 0 and 1 as function values. Variables and mathematical expressions are not needed after RND. A BAD SYNTAX error results if used in R = RND (0) or similar form.

Example:

```
>FOR I= 1 TO 5 : PRINT RND : NEXT 1
4.5777066 E-5
5.9510185 E-3
.26771954
.53534752
.55883116
```

RUN

Command

FUNCTION — Executes a BASIC-52 program.

DESCRIPTION — The RUN command clears all variables to 0 and clears interrupts. It starts execution beginning with the first line of the current program. A line number at which to start execution, such as RUN 40, cannot be designated. In this case, use a GOTO statement in the direct mode, such as GOTO 40, instead of using the RUN command.

Example:

```
READY
>LIST
10        FOR I=0 TO 4
20        PRINT EXP(I),
30        NEXT I : PRINT
40        PRINT "I=", I
50        END

READY
>RUN

 1  2.7182818  7.3890559  20.085536  54.598146
I= 5

READY
>CLEAR

>GOTO 40
I= 0

READY
>
```

Program execution can be interrupted by inputting Control-C or by the STOP statement.

SAVE

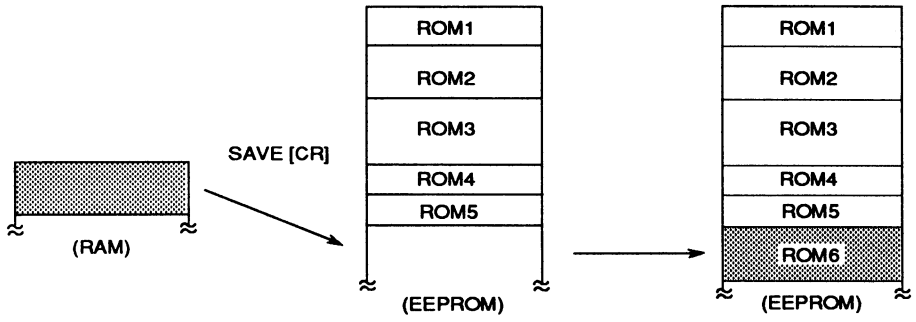
Command

FUNCTION — Saves a BASIC-52 program in the RAM to the EEPROM.

DESCRIPTION — Saves a BASIC-52 program in the RAM to the EEPROM and displays the program No. after saving.

Example:

```
>LIST
10          STRING 100, 20
20          $(0)="Hello,"
30          $(2)="My name is ASC-6210."
40          INPUT "May I have your name, please?", $(1)
50          PRINT $(0), $(1)
60          PRINT $(2)
70          PRINT : GOTO 40

READY
>SAVE
6
READY
>
```



The error messages regarding the SAVE command are:

- CAN'T SAVE PROGRAMS ANY MORE.

  There are already 255 programs saved, and no additional programs can be saved.

- MISSING OPERAND

  The SAVE command was used in the program mode.

- NO PROGRAMS ON RAM.

  There were no programs in the RAM.

- PROGRAM > RESET OF EEPROM FREE AREA.

  The program in the RAM was longer than the remaining area of the EEPROM and so was not saved.

- EEPROM MEMORY

  An error occurred during writing of BASIC-52 programs in the EEPROM.

SGN (<Mathematical Expression>)

Function

FUNCTION	The sign of the <mathematical expression> value is provided as a function value.

Example:

```
>PRINT    SGN(710),: PRINT SGN(0),: PRINT SGN(-4.5 E+09)
  1   0    -1
```

SIN (<Mathematical Expression>)

Function

FUNCTION	Provides sine (sine function)values. (y = sin x)
DESCRIPTION	Provides sine value of angle given by <mathematical expression>. The angle unit is radian. Calculations are performed in 7 digits of valid numerals. Set angles are expressed by <mathematical expression> in the range of ±200000 radian.

SPC (<Mathematical Expression>)

Function

FUNCTION — Outputs spaces for the designated value.

DESCRIPTION — Outputs the number of spaces given by the <mathematical expression>. A BAD ARGUMENT error will result if the value given by the <mathematical expression> is negative.

Example:

```
> PRINT "ASCII/BASIC", SPC(5), "module"
  ASCII/BASIC          module


>
```

SQR (<Mathematical Expression>)

Function

FUNCTION — The square root of a <mathematical expression> value is provided as a function value.

DESCRIPTION — The numerical value expressed by <mathematical expression> must be 0 or a positive number. A BAD ARGUMENT error will result if the value is negative. The result will be to a precision of the least significant numeral ±5 among the valid numerals.

Example:

```
10        PRINT "  X   SQR(X)"
20        FOR I= 0 TO 5
30        PRINT I, SPC(2), SQR(I)
40        NEXT I
50        END

READY
>RUN

X         SQR(X)
0         0
1         1
2         1.4142136
3         1.7320508
4         2
5         2.236068
```

STOP

Statement (Program)

FUNCTION — Stops program execution and returns to the direct mode.

DESCRIPTION — The STOP statement stops program execution and can be placed anywhere in a program.

Variable values can be displayed or changed when execution is stopped. Use the CONT command to resume execution beginning with the line No. at which the execution was stopped. Use a GOTO statement to resume execution by designating a line number.

Example:

```
>10     FOR I=1 TO 100
>20     PRINT I,
>30     IF I=10 STOP
>40     NEXT I
>RUN

1  2  3  4  5  6  7  8  9  10  STOP - IN LINE 40
READY
>I=90

>CONT

91  92  93  94  95  96  97  98  99  100
READY
>
```

STRING <Mathematical Expression 1>, <Mathematical Expression 2>

Statement (Direct/Program)

FUNCTION

Designates the length of a character string variable and allocates a memory area in the memory.

DESCRIPTION

A character string variable area is not allocated at start of BASIC-52. Be sure to allocate a character string variable area by this STRING statement if character string variables such as $(0)="ASCII/BASIC module" are handled in BASIC-52 programs or in the direct mode. A MEMORY ALLOCATION error will result if character string variables are handled before allocating an area by the STRING statement.

<Mathematical expression 1> and <Mathematical expression 2> represent the following:

<Mathematical Expression 1>:

The number of bytes of area allocated for character string variables.

<Mathematical Expression 2>:

The number of characters that can be set with one character string variable.

Assuming the values indicated by <mathematical expressions 1 and 2> and the number of character string variables to be T, C, and N, these three values have the following relational expression:

T=(C+1) * N+1 (bytes)

Character string variables are delimited by a carriage return (CR), and one byte is needed with each character string variable for the CR character. One byte is also needed to delimit the entire character string variable area. As a result, the foregoing relational expression establishes.

The character string variables that can be handled by BASIC-52 are linear, and the size of subscripts is maximum 254. Subscripts begin with 0 and 255 subscripts from $(0) to $(254) is the maximum number of subscripts for use as character string variables.

The character string variables declared by a STRING statement cannot be deleted by the NEW command or by a CLEAR statement. Execute the statement STRING 0, 0 during a program or in the direct mode.

The BASIC-52 interpreter processes similar to processing by a CLEAR statement for each execution of a STRING statement. This is because character string variables and numerical variables secure areas in the same external RAM. Refer to the Appendix 6. Information About ASCII/BASIC Module. By executing a STRING statement, the numerical variable data stored in the memory is cleared. Therefore, define character string variables by a STRING statement at the start of a program to avoid the numerical variable data being cleared.

Example:

```
>10   STRING 30, 12
>20   $(0)="ASCII/BASIC  "
>30   $(1)="module"
>40   PRINT $(0), $(1)
>RUN

ASCII/BASIC module

READY
>
```

If a character string is attempted to set longer than the length of character string variables declared by a STRING statement, the message 'EXTRA IGNORED' will be displayed and the portion that has surpassed the limit will not be set.

```
READY
 >STRING 20, 12 : $(0) = "BASIC-52   interpreter"
 EXTRA IGNORED

 >PRINT $(0)
 BASIC-52 int

 >
```

ST@ <Mathematical Expression>

Statement

FUNCTION

Pops the value on the argument stack and stores it at the address designated by the mathematical expression that follows ST@.

DESCRIPTION

The memory addresses expressed by the <mathematical expression> must be ones that are effective inside the ASCII module and that do not destroy work areas of the ASCII module and of BASIC-52. Effective addresses are addresses 205H to MTOP (normally address 73FFH). Be sure to push values to the A-STACK before executing this command. An A-STACK error will result if this command is executed without pushing values to the A-STACK.

Use the ST@ command in combination with the LD@ command.

TAB (<Mathematical Expression>)

Function

FUNCTION

Outputs spaces to the designated position on the line in which the cursor is located.

DESCRIPTION

The decision on the position of output for data to be output next is made in accordance with the value given by <mathematical expression>. If the value of <mathematical expression> is equal to or less than the present position of the cursor or print head, the TAB function is ignored.

A BAD ARGUMENT error will result if the value given by the <mathematical expression> is negative.

Example:

```
>LIST
10        FOR I=0 TO 9
20        PRINT TAB (I), "ASCII/BASIC", TAB (I+20), "module"
30        NEXT I

READY
>RUN

ASCII/BASIC          module
 ASCII/BASIC          module
  ASCII/BASIC          module
   ASCII/BASIC          module
    ASCII/BASIC          module
     ASCII/BASIC          module
      ASCII/BASIC          module
       ASCII/BASIC          module
        ASCII/BASIC          module
         ASCII/BASIC          module

READY
>
```

TAN (<Mathematical Expression>)

Function

FUNCTION — Provides tangent (tangent function) values. (y=tan x)

DESCRIPTION — Provides tangent value of angle given by <mathematical expression>. The angle unit is radian. Set angles are expressed by <mathematical expression> in the range of ±200000 radian.

TIME

Special Function

FUNCTION — Reads and sets real-time clock values in BASIC-52 .

DESCRIPTION — The TIME function value is set to 0 when BASIC-52 is started. Executing the CLOCK 1 statement switches on the real-time clock. The TIME function value increases every 5 msec after the real-time clock switches on. The TIME function unit is seconds.

The TIME function usesTIMER0 and interrupts created in conjunction with TIMER0 on 8052AH. An appropriate value must be set as the XTAL value to make the TIME function value accurate.

A value can be set with the TIME function. For example, however, if TIME=50 is set, only the integer part of the TIME function can be set. In order to set a decimal part of the TIME function, set address 47H (71) in the internal memory to DBY (47H)=n. The decimal part is set to the value of (5 msec * n). An integer value between 0 and 255 (0FFH) is valid as the value of n.

Example:

```
READY
>PRINT TIME
0
```
The TIME function value is 0 when BASIC-52 is started.

```
READY
>CLOCK1
```
Switches the real-time clock on.

```
READY
>CLOCK0
```
Switches the real-time clock off.

```
READY
>PRINT TIME, DBY(47H)
5.035 7
```
Displays the time that the real time clock was on. The 0.035 sec is for the time below the decimal point.

```
READY
>DBY(47H)=120:  PRINT TIME
5.6
```
Sets time below the decimal point to 0.005 * 120=0.6 sec.

TIMER0

Special Function

FUNCTION — Reads values of TH0 and TL0 registers of 8052AH.

DESCRIPTION — TIMER0 is a function with a 16 bit value storing values of data registers TH0 and TL0 corresponding to timer/counter 0. Therefore, the value of it changes with each reading. The BASIC-52 interpreter uses TIMER0 when the real time clock is activated.

TIMER1

Special Function

FUNCTION — Reads values of TH1 and TL1 registers of 8052AH.

DESCRIPTION — TIMER1 is a function with a 16 bit value storing values of data registers TH1 and TL1 corresponding to timer/counter 1. Therefore, the value of it changes with each reading. The ASCII module uses this value in setting the baud rate of the CH2 serial port.

TIMER2

Special Function

FUNCTION — Reads values of TH2 and TL2 registers of 8052AH.

DESCRIPTION — TIMER2 is a function with a 16 bit value storing values of data registers TH2 and TL2 corresponding to timer/counter 2. The ASCII module uses this value to set the baud rate of the CH1 serial port.

USING (Format Control Character String)

Function

FUNCTION — Outputs values after converting them into the designated format. U. can be used as a short form of the USING function.

DESCRIPTION — The BASIC-52 interpreter saves the format if a USING function is executed once. Thus, values output subsequently are output in the format designated by a USING function. The following formats can be set by a USING function:

USING (Fx): By setting to this format, all values will be output as floating-decimal numbers. The 'x' indicates the number of effective digits. Integers between 0 and 8 are valid as the range of 'x'.

Example:

```
>LIST
10      PRINT USING (F0), 0, 1, 2
20      PRINT USING (F1), 0, 1, 2
30      PRINT USING (F2), 0, 1, 2
40      PRINT USING (F3), 0, 1, 2

READY
>RUN

0.0 E 0 1.0 E 0 2.0 E 0
0.00 E 0 1.00 E 0 2.00 E 0
0.00 E 0 1.00 E 0 2.00 E 0
0.00 E 0 1.00 E 0 2.00 E 0

READY
>
```

USING(#.#): By setting to this format, all values will be output in decimals format. The period (.) indicates decimal point. The number of #'s in front of the decimal point indicates the number of digits of the integer part of values to be output.

The number of #'s following the decimal point indicates the number of digits of the decimal part of values to be output. The decimal point can be omitted, in this case, only the integer part of values will be output. A maximum of eight #'s are permitted.

A '?' (question mark) will be output in front of a value, and values will be output in a free format if output in the set form is not possible because the value is too large or for other reasons. Refer to the next paragraph for the free format.

Example:

```
>LIST
10        PRINT USING (##.###), 0,1,2
20        FOR I=0 TO 5
30        PRINT I**3
40        NEXT I

READY
>RUN

0.000   1.000   2.000
0.000
1.000
8.000
27.000
64.000
? 125

READY
>
```

USING(0): This format is set when BASIC-52 is started. If the value is between ±0.1 and ±99999999, values are output by mixing integers and decimals. Values are output in the USING (F0) format if values are outside this range. The 0's at the top and end of values, such as values are less than 1 are always omitted.

CAUTION

Do not type a space between "USING" and a right parenthesis "(" when creating a program utilizing a USING function.

Example: Correct: - PRINT USING(###.##), A
Wrong: - PRINT USING (###.##), A

XBY (<Mathematical Expression>)

Function

FUNCTION — Reads and sets data of the external data memory of the ASCII module.

DESCRIPTION — Reads data of the external data memory of the ASCII module and sets values.

Values between 0 and 65535 (0FFFH) are valid as the range of values expressed by <mathematical expression>. Values between 0 and 255 (0FFH) are valid as set values. If values are outside this range, a BAD ARGUMENT error will occur.

CAUTION The ASCII module may operate incorrectly if the value of any address of the external data memory is set with an XBY ( ) function. Do not set values other than in the area shown by "ASCII Module Memory Layout", in APPENDIX 6 INFORMATION ABOUT THE ASCII MODULE.

Example:

```
>PRINT XBY(7408H)
 15
>
```

XFER

Command

FUNCTION

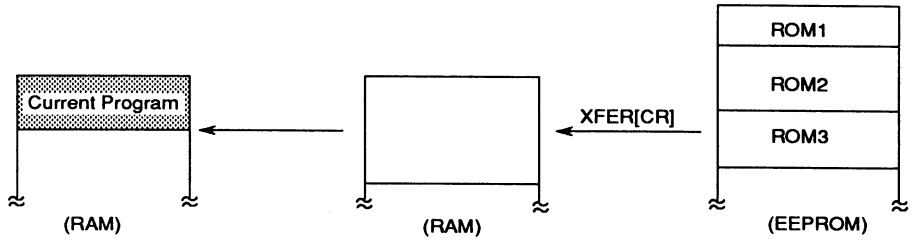
Transfers currently selected program to the RAM and sets BASIC-52 operation to RAM mode.

DESCRIPTION

Transfers the program currently selected as the current program to the RAM and sets BASIC-52 operation to the RAM mode. In order to edit the program saved in the EEPROM that is selected by the PROM command, first transfer it to the RAM using this command, and then edit the program.

If a program already exists in the RAM, it clears the program in the RAM beforehand to fransfer the program in the EEPROM to the RAM using this XFER command.

Example:



Executing the XFER command causes no changes if the program already existing in the RAM is selected as the current program.

XTAL

Special Function

FUNCTION

Reads and sets the frequency value used by BASIC-52.

DESCRIPTION

The XTAL value is used by the BASIC-52 interpreter for calculation of real-time clock values and baud rates of the software serial port (CH1 of the ASCII module). The unit of the XTAL value is Hz [hertz].

The ASCII module sets the XTAL value = 12 MHz. Do not change this value.

5.4 Built-in Subroutines

The ASCII module has built-in ASSEMBLER subroutines utilizing BASIC-52 CALL statements. These subroutines are used when the ASCII module interfaces with the EX CPU (CALL 17 excepted). Interfacing is made easy by CALLing these subroutines during a BASIC-52 program.

Built-in subroutines are divided roughly into three types:

- Dedicated data type conversion routines
- Dedicated interface flag processing routines
- Other

First, the built-in subroutines are listed, and then the individual subroutines are described in details. Dedicated data type conversion routines.

5.4.1 List of built-in subroutines

Dedicated data type conversion routines

No. During CALL	For WRITE/READ Processing	Processing
0	WRITE	Fetches one set of text data sent by EX CPU from the buffer memory and stores it in $(0) of character string variable storage area.
2	WRITE	Fetches one integer data (0≤ n≤65535) in positive word length sent by EX CPU from the buffer memory and pushes it to A-STACK.
3	WRITE	Fetches one data (0 ≤n≤4294967295) in positive double-word length sent by EX CPU from the buffer memory and pushes it to A-STACK.
6	WRITE	Fetches one signed word-length integer data sent by EX CPU (-32768 ≤ s≤32767) from the buffer memory and pushes it to A-STACK.
7	WRITE	Fetches one signed double-word length integer data (-2147483648≤s≤2147483647) sent by EX CPU from the buffer memory and pushes it to A-STACK.
8	READ	Fetches one set of character string variables set to $(0) during execution of a BASIC-52 program and writes it in the buffer memory.
10	READ	Pops one signed integer data (-32768≤s≤ 32767) pushed to A-STACK during a BASIC-52 program and writes it in the buffer memory.
11	READ	Pops one signed integer data (2147483648≤s≤2147483647) pushed to A-STACK during a BASIC-52 program and writes it in the buffer memory.
18	WRITE	Pushes the number of data words sent by EX CPU to A-STACK.
19	READ	Declares during a BASIC-52 program that READ data will now be set in the buffer memory.

Dedicated Interface Flag Processing Routines

No. During CALL	For WRITE/READ Processing	Processing
13	READ	Sets the READ data set end device to '0' and READ acknowledge enable device to '1' and then notifies EX CPU that the ASCII module has finished the BASIC-52 routine for READ processing.
14	WRITE	Sets the WRITE acknowledge enable device to '1' and then notifies EX CPU that the ASCII module has finished the BASIC-52 routine for WRITE processing.
15	WRITE (Application)	Sets the READ data set end device to '0' and READ acknowledge enable and WRITE acknowledge enable devices to '1'. Then notifies EX CPU that the ASCII module has finished the BASIC-52 routine for WRITE with ANSWER processing.
16	Both READ and WRITE	Readies for READ and WRITE processing commands to be received from EX CPU.
20	READ	Sets the READ data set end device to '1' and READ acknowledge enable device to '0' and then notifies EX CPU that the ASCII module has finished setting READ data.

Other

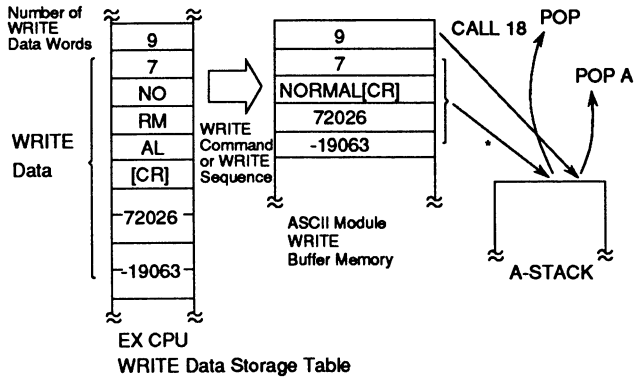
No. During CALL	For WRITE/READ Processing	Processing
17	————	Executes the BASIC-52 program of the program No. pushed to A-STACK.

5.4.2 Dedicated data type conversion routine

The dedicated data type conversion routine sends data from the EX CPU to the ASCII module (WRITE processing). It also sends data from the ASCII module to the EX CPU (READ processing). This CALL subroutine is needed, because the data handling methods in the EX CPU and in the ASCII module differ. The data flows and CALL subroutines during data writing and reading by the EX CPU are illustrated below.

• During Writing

The following explanation presents an example of writing four types of data—7, 'NORMAL', 72026 and -19063 from the EX CPU to the ASCII module. These four types of data are stored in the WRITE data storage table of the EX CPU as shown below. The CALL subroutine is operated as follows using the four types of data in BASIC-52 programs after the EX CPU executes the WRITE sequence/WRITE command and the BASIC-52 program in the ASCII module is started.




```
10 REM *** EX WRITE ***
20 STRING 50, 10
30 CALL 18 : POP A
40 PRINT "WORD SIZE =", A
50 CALL 2 : POP B
60 CALL  0 : $(1)=$(0)
70 PRINT B, "LINES_", $(1)
80 CALL 3 : POP C
90 PRINT C, "COUNTS"
100 CALL 7 : POP D
110 PRINT " SPEED   :", D
 :     :
 :     :
```

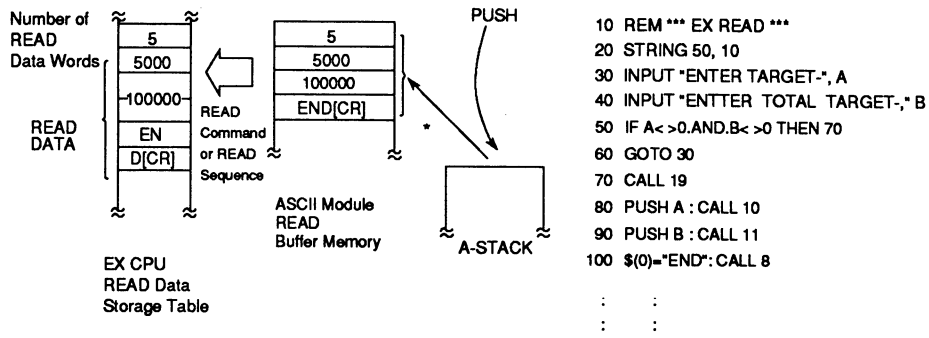
* Subroutines for calling differ with WRITE data in accordance with data type. A-STACK is not used for text data, but text data is processed by nearly the same method. Therefore, processing of text data is also described below.

The CALL subroutines for WRITE processing push data written by the EX CPU to A-STACK. (Text data is stored in $(0).) This enables the use of data in BASIC-52 programs.

After calling these subroutines, be sure to fetch data from the A-STACK using the BASIC-52 POP statement. (Transfer text data to an area other than $(0) by using LET $(1) = $(0) or by other means.) Continuing the subroutine CALL and POPping later in a batch causes stack overflow of BASIC-52 programs and disrupts BASIC-52 program execution. Use subroutine calls and the POP statement as a pair.

• During Reading

The following explanation presents an example of sending three types of data—5000, 100000, and 'END'— from the ASCII module to be read by the EX CPU.
The EX CPU executes the READ sequence/READ command to start the ASCII module BASIC-52 program. If data to be sent to the EX CPU is obtained during the BASIC-52 program, operate the CALL subroutine as illustrated below to write the three types of data in the buffer memory.



- Subroutines for calling differ for READ data in according to the data type. A-STACK is not used for character string variables (text data). However, the processing method for them is nearly the same and is also described below.

The CALL subroutines for READ processing fetch data on the A-STACK and write it in the buffer memory. (Character string variables are directly written stored in $(0) without using the A-STACK.) By writing data in the buffer memory, the EX CPU is assured of data feed. Therefore, be sure to PUSH data to the A-STACK before calling these subroutines during BASIC-52 program execution. (In the case of text data, transfer data to $(0) by changing LET $(0)=$(1) or by other method.) Continuing data PUSHing and CALLing subroutines in batch later during execution of BASIC-52 programs causes stack overflow and disrupts BASIC-52 program execution. Use the PUSH statement and subroutine calls in pair.

CALL 0

Processing Mode: WRITE

FUNCTION — Fetches one text data sent by the EX CPU from the buffer memory and stores it at $(0) of the character string variable storage area.

DESCRIPTION — Provides a character string variable storage area during the BASIC-52 program by the STRING statement when this subroutine is used. CALLing this subroutine without

executing the STRING statement causes misoperation of the ASCII module.

Use the [CR] code as a data delimiter when setting text data in the EX CPU.

Exmple: When three types of text data are sent by the EX CPU, namely, "NORMAL", "STOP" AND "END."

H 4E 4F
H 52 4D
H 41 4C
H 0D 20
H 53 54
H 4F 50
H 0D 20
H 45 4E
H 44 0D

EX CPU

WRITE Data Storage Table

CALL 2/CALL 3

Processing Mode: WRITE

FUNCTION — Fetches one positive integer data sent by the EX CPU from the buffer memory and PUSHes it to the A-STACK.

DESCRIPTION — The range of data that can be handled is as follows:
CALL2: $0 \leq n \leq 65535$
CALL3: $0 \leq n \leq 4294967295$

CALL 6/CALL 7

Processing Mode: WRITE

FUNCTION — Fetches one signed integer data sent by the EX CPU from the buffer memory and PUSHes it to the A-STACK.

DESCRIPTION — The range of data that can be handled is as follows:
CALL 6: $-32768 \leq s \leq 32767$
CALL 7: $-2147483648 < s < 2147483647$

CALL 8

Processing Mode: READ

FUNCTION — Fetches one set of character string variables set at $(0) during the BASIC-52 program and writes it in the buffer memory as text data.

DESCRIPTION — In order to use this subroutine, secure a character string variable area by the STRING statement and set effective data in $(0) of the character string variable during the BASIC-52 program. CALLing this subroutine without executing these steps causes misoperation of the ASCII module, and abnormal data will be sent to the EX CPU.

CALL 10/CALL 11

Processing Mode: READ

FUNCTION — POPs one signed integer data pushed to the A-STACK during a BASIC-52 program and writes it in the buffer memory.

DESCRIPTION — The range of data that can be handled is as follows:
CALL 10: $-32768 \leq s \leq 32767$
CALL 11: $-2147483648 \leq s \leq 2147483647$

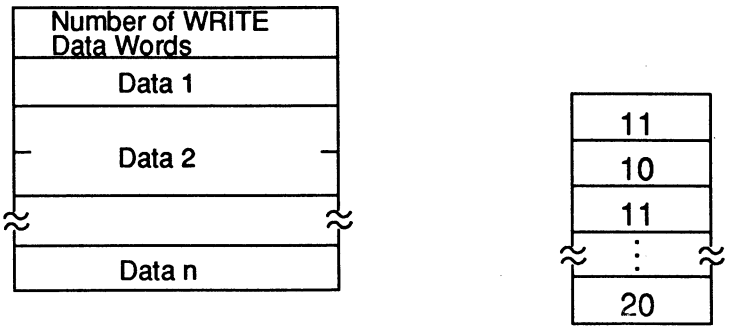
CALL 18

Processing Mode: WRITE

FUNCTION — PUSHes the number of words of data sent by the EX CPU to the A-STACK.

DESCRIPTION — The configuration of the data sent by the EX CPU to the ASCII module (written) is as follows:

Example: When sending 11 pieces of integer data from the EX CPU, namely, Data 10 to 20.



Therefore, in this case, (number of WRITE data words) = (number of WRITE data). The number of WRITE data that can be obtained by CALLing this subroutine can be utilized in the BASIC-52 programs.

CALL 19

Processing Mode: READ

FUNCTION — Declares in BASIC-52 programs that READ data will now be set in the buffer memory.

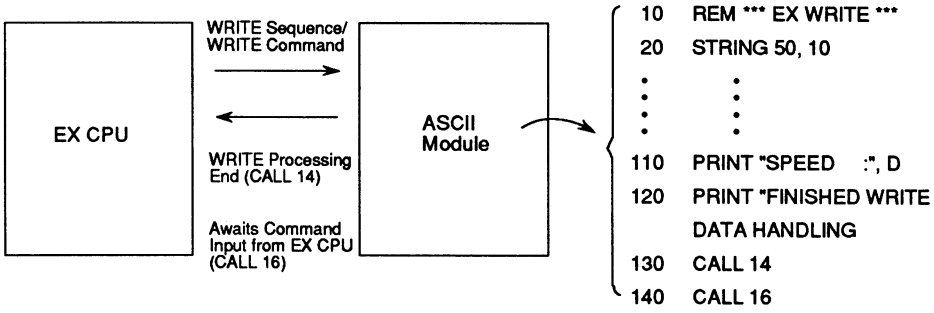
DESCRIPTION — Be sure to CALL this subroutine before writing data to be sent to the EX CPU in the buffer memory by CALLing the subroutine during a BASIC-52 program started by the READ sequence/READ command. Data may not be sent correctly if it is sent to the EX CPU without CALLing this subroutine.

5.4.3 Dedicated interface flag processing routine

This routine is used to notify the EX CPU that the ASCII module has set READ data and that the processing of READ sequence/READ command or WRITE sequence/WRITE command has been finished.
The CALL subroutines and timing used in data writing and reading by the EX CPU are illustrated below.

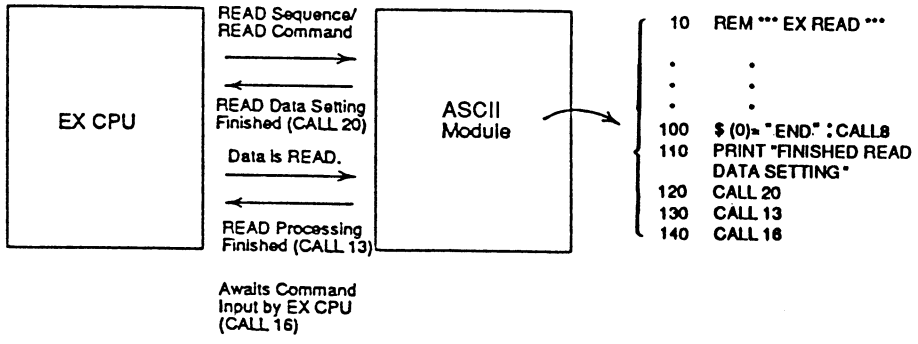
• During Writing

Data is used in BASIC-52 programs utilizing the CALL subroutine for data type conversion when the EX CPU executes the WRITE sequence/WRITE command and a BASIC-52 program in the ASCII module starts.
After BASIC-52 processing finishes using data written by the EX CPU, the flag processing routine is called to notify the EX CPU of the condition of the ASCII module and other information.



• During Reading

The EX CPU executes the READ sequence/READ command to start the BASIC-52 program by the ASCII module.
After data to be sent to the EX CPU has been obtained in the BASIC-52 program, the data is written in the buffer memory using the CALL subroutine.
The flag processing routine is CALLed when the EX CPU is ready to read the data after all the data to be read by the EX CPU is written in the buffer memory.



CALL 13

Processing Mode: READ

FUNCTION

Sets the READ data set end device to '0' and READ acknowledge enable device to '1' and notifies the EX CPU that the ASCII module has finished the BASIC-52 routine for READ processing.

DESCRIPTION

This subroutine is called after notifying the EX CPU that data to be read has been set by executing CALL 20 during a BASIC-52 program. The ASCII module notifies the EX CPU that all processing of data for reading by the EX CPU has been finished.

CALL 14

Processing Mode: WRITE

FUNCTION

Sets the WRITE acknowledge enable device to '1' and notifies the EX CPU that the ASCII module has finished the BASIC-52 routine for WRITE processing.

DESCRIPTION

This routine is called to notify the EX CPU by the ASCII module that all processing of the data written by the EX CPU during BASIC-52 programs has been finished.

CALL 15

Processing Mode: WRITE (Application)

FUNCTION

Sets the READ data set end device to '0' and the READ acknowledge enable and WRITE acknowledge enable devices to '1'. Also notifies the EX CPU that the ASCII module has finished the BASIC-52 routine for WRITE with ANSWER processing.

DESCRIPTION

This routine is called to notify the EX CPU by the ASCII module that all processing regarding WRITE with ANSWER during the BASIC-52 program has been finished. WRITE with ANSWER is a WRITE processing application. For specific examples, refer to section 6.3 WRITE with ANSWER processing in "CHAPTER 6. PROGRAMMING APPLICATIONS."

CALL 20

Processing Mode: READ

FUNCTION — Sets the READ data set end device to '1' and READ acknowledge enable devices to '0' and notifies the EX CPU that the ASCII module has finished READ data setting.

DESCRIPTION — This subroutine is called after all data for the EX CPU to read utilizing the data conversion subroutine has been written in the buffer memory and when the EX CPU is ready to read the data.

CALL 16

Processing Mode: WRITE/READ

FUNCTION — Readies the ASCII module to accept the READ processing/WRITE processing command from the EX CPU.

DESCRIPTION — Executes CALL 13, 14, 15, and others during BASIC-52 program execution and notifies the EX CPU that all processing regarding READ and WRITE has been finished. By calling this subroutine, the ASCII module will again be able to accept the READ and WRITE processing command from the EX CPU.

CAUTION — Executing the CALL 16 statement readies the ASCII module to receive commands from the EX CPU, disabling it to accept all BASIC-52 commands. The only way to stop processing the CALL 16 statement is to press the RESET switch on the ASCII module for resetting. Pay careful attention to this during BASIC-52 program creation and debugging.

5.4.4 Others

Regardless of whether or not the EX CPU and ASCII module are interfaced the following command can be used as one of the BASIC-52 commands.

CALL 17

FUNCTION

POPs program No. pushed to the A-STACK and automatically executes the BASIC-52 program of the No. that have been POPped.

DESCRIPTION

This command combines the functions of the PROM and RUN commands. This is a CALL statement and can execute both in the direct and program modes. Executing the CALL 17 statement by PUSHing an invalid program No. will cause the CALL 17 statement to ignore the No. and refuse processing. The statements in the following lines of the program are executed.

Example

```
> 10 INPUT "ENTER ROM NO.- ",A
> 20 PUSH  A
> 30 CALL  17
> RUN

ENTER ROM NO.- 1 [CR]
*** ROM 1 PROGRAM ***
20 PROGRAMS ARE IN EEPROM.

READY
>
```

5.5 BASIC-52 Error Messages

There are two ASCII module conditions during BASIC-52 program editing and debugging. The one condition is the BASIC-52 program execution condition (program mode). The other condition enables BASIC-52 program editing and BASIC-52 command direct input (direct mode). The indication method differs between the program mode and direct mode even if the error message is the same.

In the program mode, the line in which the error has occurred is indicated together with the error message as follows:

```
ERROR: BAD SYNTAX - IN LINE 20
20      A=A1+A2- : PRINT A
- - - - - - - - - - - - - x
```

In the direct mode, only the error message is displayed as follows. (In RENUM only, the number of the line in which the error has occurred is also indicated.)

```
A=123+456- : PRINT A [CR]
ERROR: BAD SYNTAX
```

Error messages and troubleshooting are shown in the following pages in an alphabetical order.

List of Error Messages

Indicated Message	Message Meaning	Mode	Troubleshooting
ERROR: ARITH. OVERFLOW	Overflow has occurred as a result of mathematical operations.	Direct/ program	Make sure that variables do not become larger than ±.99999999E+127.
ERROR: ARITH. UNDERFLOW	Underflow has occurred as a result of mathematical operations.	Direct/ program	Make sure that variables do not become smaller than ±1E-127.
ERROR: ARRAY SIZE	Attempt to access a variable that surpasses range of array variables set by DIM statement .	Direct/ program	Check if sentences or lines containing array variables are within range set by DIM statement.
ERROR: A-STACK	Data pushed beyond A-STACK range, or popped even though there was no data in A-STACK.	Direct/ program	Check that push and pop are used in a pair, or if more than 30 pieces of data are pushed at one time.
ERROR: BAD ARGUMENT	Variables used in operator surpassed operator variable range.	Direct/ program	Check that variables used in operator are within range .
ERROR: BAD SYNTAX	Program is not grammatically correct.	Direct/ program	Check that the program is grammatically correct.
ERROR: CAN'T CONTINUE	Programs cannot be continued by CONT command.		The following three cases are possible: 1) Tried to CONTinue the program stopped by an error. 2) Tried to CONTinue corrected and edited programs during interruption of program execution. 3) Tried to CONTinue a program that does not exist. Check if the program tried to be continued fit these three cases.
ERROR: CAN'T SAVE PROGRAMS ANY MORE	255 programs are already saved in EEPROM . Additional programs cannot be saved.	Direct	Kill unnecessary programs or save programs on a floppy disk to back up and sort EEPROM memory contents.

Indicated Message	Message Meaning	Mode	Troubleshooting
ERROR: C-STACK	Too many statements requiring looping such as FOR-NEXT and DO statements were used in nesting. Or, RETURN, WHILE, UNTIL, or NEXT statement in GOSUB-RETURN, DO, or FOR-NEXT statements is placed befor GOSUB, DO or FOR statement.	Direct/ program	Check that program loop processing parts and parts using subroutines are grammatically correct, and that they are not overused in nesting.
ERROR: DIVIDE BY ZERO	Division was performed using 0 as a divisor.	Direct/ program	Check for divisor of 0 in sentences or lines that use division.
ERROR: EEPROM MEMORY	A BASIC-52 program write error has occurred with the EEPROM during SAVE/INS/KILL processing.	Direct	Check if the PRG. EN switch of the DIP switch on the front panel is set to program write enable (ON).
ERROR: INVALID LINE NO. IN xxx	Error was found in line number of line number reference statements such as GOTO and GOSUB statements when RENUM command was executed.	Direct	Check that all line numbers in line number reference statements of lines shown by xxx are correct.
ERROR: INVALID LINE NUMBER IN xxx	Line number used in line number reference statements such as GOTO and GOSUB statements did not exist.	Program	Check that all line numbers in line number reference statements of lines shown by xxx are correct.
ERROR: INVALID PARAMETER	Value input as parameter was abnormal	Direct	Check that value input as parameter is within effective range.
ERROR: INVALID ROM NO.	Designated ROM program No. was invalid during PROM/INS/KILL command execution.	Direct	The following is possible. Designated program No. was not within valid range ($0 < n \leq$ (number of programs presently existing), where n is an integer). Count the ROM programs and confirm.

Indicated Message	Message Meaning	Mode	Troubleshooting
ERROR: I-STACK	Internal stack area needed to evaluate mathematical expressions is not large enough.	Direct/ program	This error does not occur if BASIC-52 is used normally. If this error occurs, press the RESET switch to reset the ASCII module. If this error occurs even after resetting, internal CPU trouble is probable. Contact our service department or agent.
ERROR: LIMIT OVER LINE NO.	Some lines show a line number of 65536 or larger if RENUM command is executed.	Direct	Correct the BASIC-52 program to prevent lines with a line number larger than 65535 from occurring after executing RENUM command.
ERROR: MEMORY ALLOCATION	Valid range set as string was surpassed, or a value unsuitable as MTOP of system variable was set.	Direct/ program	Process as follows: 1) Check that statements and lines containing a string are within the range set by STRING statement. 2) Check if '29695' is indicated as MTOP if PRINT MTOP [CR] is input. If MTOP value is other than 29695, check if MTOP value was rewritten in program after keying MTOP = 29695 [CR].
ERROR: MISSING OPERAND	Statements that cannot be used in program mode exist in the BASIC-52 program.	Program	Check if statements that can only be used in direct mode such as PROM and SAVE exist in the program.
ERROR: NO DATA	Data read in READ statements are not available in DATA statements, or number of data is not sufficient.	Program	Check if data is correctly set in DATA statements and if data is available in quantity larger than that read in READ statements.
ERROR: NO PROGRAMS ON RAM.	SAVE/INS was tried even though RAM had no program.	Direct	Check that a program exists in RAM.

Indicated Message	Message Meaning	Mode	Troubleshooting
ERROR: PRO-GRAM> REST OF EEPROM FREE AREA	The program in RAM is longer than remainder of EEPROM memory, and SAVE/INS processing was not performed.	Direct	Kill unnecessary programs or save programs on a floppy disk to back up and sort EEPROM memory contents.
ERROR: UNDE-FINED LINE NO. IN xxx	Line numbers of line number reference statements such as GOTO and GOSUB statements were non-existent when RENUM command was executed.	Direct	Check that line numbers in line number reference statements shown by xxx are line numbers existing in the program.

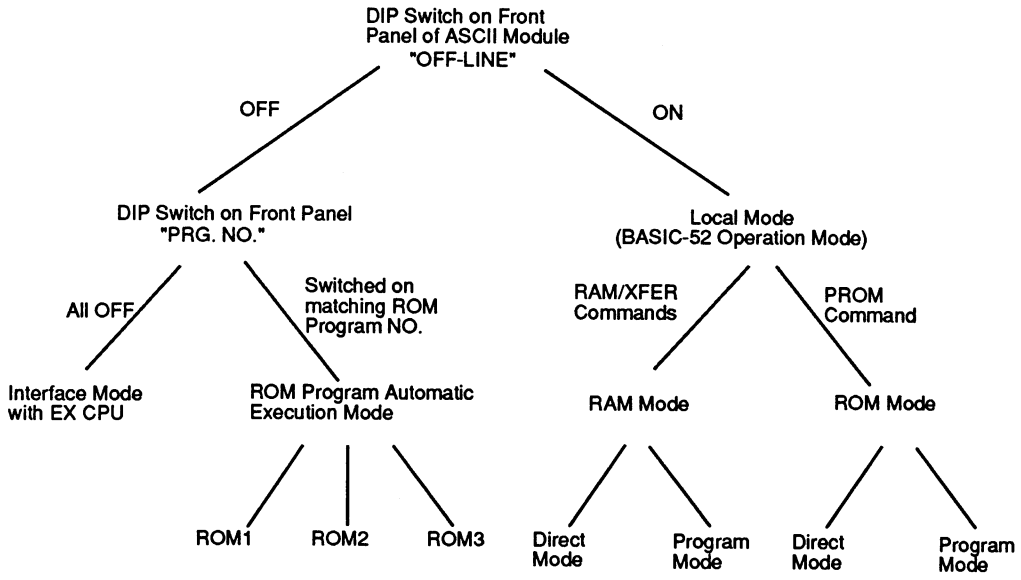
6. Information About the ASCII/BASIC Module

This appendix describes information regarding the ASCII module that may be useful during operation.
The following items are described.

- ASCII Module Mode Transition Diagram
- ASCII Module Memory Layout

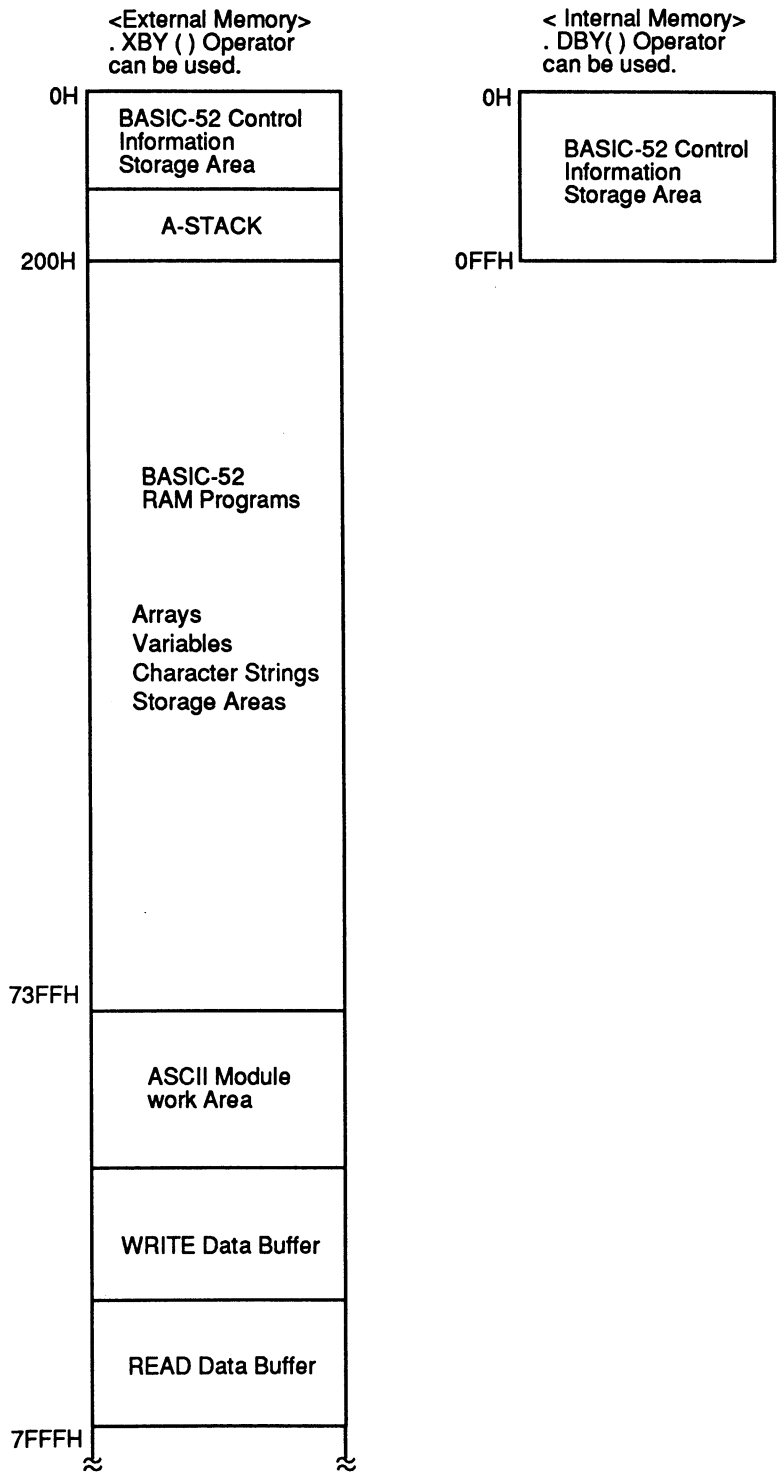
ASCII Module Mode Transition Diagram

By setting the DIP switch installed on the front panel, you can make selection from among several modes of the ASCII module. Various modes also exist in the description of the BASIC-52 language. These modes are summarized below.



ASCII Module Memory Layout

The points that are necessary for BASIC-52 program creation, editing, and debugging are described below.



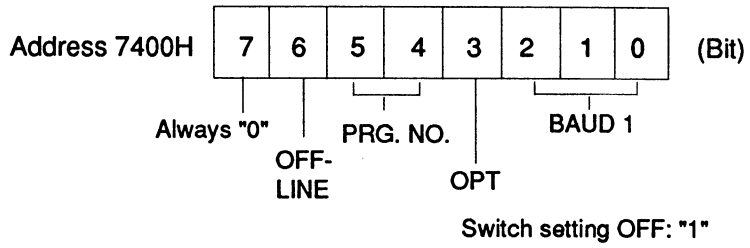
The detailed information of the external and internal memory storage areas is given below.

- **External Memory**

<table>
<tr><th>Address</th><th>Function</th></tr>
<tr><td>101H</td><td>This is the area to store error codes. The nature of the error that has occurred can be known by reading this address in the ONERR statement of the BASIC-52 program.</td></tr>
<tr><td>12DH ~
1FEH</td><td>Argument Stack (A-STACK). Used when data is exchanged between BASIC-52 and built-in subroutines.</td></tr>
<tr><td>73FFH</td><td>Last address of external memory area allocated for BASIC-52 programs. Address read by PRINT MTOP is this address.</td></tr>
<tr><td>7400H</td><td>Stores the flag that shows the ASCII module condition during interface between the ASCII module and EX CPU. By reading this address, the interface status can be known. Specific cases of the flags are shown below. Refer to CHAPTER 6. PROGRAMMING APPLICATIONS for information on how to operate these flags. Only read the data of this address. Do not write.

Address 7400H
<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>
(Bit)

<table>
<tr><th>Bit</th><th>Function</th></tr>
<tr><td>0</td><td>Set to "1" if the EX CPU can read the data.
Save the status if the EX CPU is reading the data.
Set to "0" when EX CPU finishes reading data.</td></tr>
<tr><td>1</td><td>Set to "1" if the ASCII module can accept the READ sequence/READ command.</td></tr>
<tr><td>2</td><td>Set to "1" when the EX CPU finishes reading data.</td></tr>
<tr><td>3</td><td>Set to "1" when the EX CPU starts read processing.</td></tr>
<tr><td>4</td><td>Set to "1" if the ASCII module can accept the WRITE sequence/WRITE command.</td></tr>
<tr><td>5</td><td>Set to "1" when the EX CPU finishes data writing.</td></tr>
<tr><td>6</td><td>This No. is reserved.</td></tr>
<tr><td>7</td><td>Set to "1" when the watchdog timer of the ASCII module is operating.</td></tr>
</table>
</td></tr>
</table>

Address	Function
7408H	The value that indicates the number of BASIC-52 programs stored in the EEPROM memory is stored.
7700H	The DIP switch settings on the front panel can be known by reading the data of this address in the form of A=XBY (7700H).  The data of Address 7700H and DIP switch settings correspond as illustrated above. By using this data, the OPT switch information can be used in the BASIC-52 programs.

- **Internal Memory**

Address	Function
15H	The value set by the NULL command is stored here. Thus, executing the DBY(15H) = 100 statement is the same processing as that for executing NULL 100.
47H	Of special function TIME values, values after the decimal point are stored. If 30 is stored at this address, the value of the real-time clock after the decimal point will be 0.005*30 = 0.15, that is, 0.15 second.

A

B

C

D

E

F

G

H

Q

R

S

T

U

V

W

X