MS8604A
Digital Mobile Radio Transmitter Tester

Operation Manual

Vol. 3

(Personal Test Automation
PTA / PTL - V1.5)

First Edition

Measuring Instruments Division
Measurement Group

ANRITSU CORPORATION

NOV.
1994

MS8604A Digital Mobile Radio
Transmitter Tester
Operation Manual
Vol. 3
(Personal Test Automation
PTA / PTL - V1.5)

December 1993 (First Edition)

The contents of this manual may be changed without prior notice.

Printed in Japan

WARNING

- *The protective earth terminal of this instrument must be connected to ground. The three-core power cord supplied with the instrument can be plugged into a grounded two pole AC outlet. If no grounded two pole AC outlet is available, the ground pin of the power cord or the earth terminal on the rear panel must be connected to ground before supplying the power to the instrument. Failure to do so could cause dangerous or possibly fatal electric shocks.*
- *Replacing fuses with the power cord still plugged into an AC outlet could also cause electric shocks.*
- ***Supplemental explanation about WARNING on the rear panel***

WARNING ⚠
NO OPERATOR SERVICEABLE PARTS INSIDE.
REFER SERVICING TO
QUALIFIED PERSONNEL.

A supplemental explanation about the WARNING labeled on the rear panel is given in the following:

Disassembly, adjustment, maintenance, or other access inside this instrument by unqualified personnel should be avoided. Maintenance of this instrument should be performed only by Anritsu trained service personnel who are familiar with the risks involved of fire and electric shock. Potentially lethal voltages existing inside this instrument, if contacted accidentally, may result in personal injury or death, or in the possibility of damage to precision components.

■ SAFETY CONSIDERATIONS:

Anritsu uses the following labels to identify safety precautions which should be followed to prevent personal injury or product damage. Please familiarize yourself with them before operating this product.

Labels used in this manual:

WARNING : Indicates that the procedure could result in personal injury if not correctly performed. Do not proceed before you fully understand the explanation given with this symbol and meet the required conditions.

CAUTION : Indicates that the operating procedure could result in damage to the product if not correctly performed. Do not proceed before you fully understand the explanation given with this symbol and meet the required conditions.

Note : Indicates that information helpful in understanding the operation of the product is about to be presented.

Labels or symbols used on/in the product:

⚠ : This international caution symbol indicates that the operator should refer to the operation manual before beginning a procedure.

⏚ : This symbol indicates an earth (ground) terminal. The product should be grounded via the earth terminal if a three prong power cord is not used.

CERTIFICATION

ANRITSU CORPORATION certifies that this instrument has been thoroughly tested and inspected, and found to meet published specifications prior to shipping.
Anritsu further certifies that its calibration measurements are based on the Japanese Electrotechnical Laboratory and Radio Research Laboratory standards.

WARRANTY

All parts of this product are warranted by Anritsu Corporation of Japan against defects in material or workmanship for a period of one year from the date of delivery.
In the event of a defect occurring during the warranty period, Anritsu Corporation will repair or replace this product within a reasonable period of time after notification, free-of-charge, provided that: it is returned to Anritsu; has not been misused; has not been damaged by an act of God; and that the user has followed the instructions in the operation manual.

Any unauthorized modification, repair, or attempt to repair, will render this warranty void.

This warranty is effective only for the original purchaser of this product and is not transferable if it is resold.

ALL OTHER EXPRESSED WARRANTIES ARE DISCLAIMED AND ALL IMPLIED WARRANTIES FOR THIS PRODUCT, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO A PERIOD OF ONE YEAR FROM THE DATE OF DELIVERY. IN NO EVENT SHALL ANRITSU CORPORATION BE LIABLE TO THE CUSTOMER FOR ANY DAMAGES, INCLUDING LOST PROFITS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE THIS PRODUCT.

All requests for repair or replacement under this warranty must be made as soon as possible after the defect has been noticed and must be directed to Anritsu Corporation or its representative in your area.

'IBM' is a registered trademark of the IBM Corporation.
'HP' is a registered trademark of the Hewlett-Packard Company.
'MS-DOS' is a registered trademark of the Microsoft Corporation.
'NEC' is a registered trademark of the NEC Corporation.

MEMORY BACK-UP BATTERY REPLACEMENT

The power for memory back-up is supplied by a Poly-carbomonofluoride Lithium Battery. This battery should only be replaced by a battery of the same type; since replacement can only be made by Anritsu, contact the nearest Anritsu representative when replacement is required.

STORAGE MEDIUM

This equipment stores data and programs using Plug-in Memory cards (PMC) and backed-up memories. Data and programs may be lost due to improper use or failure. ANRITSU therefore recommends that you back-up the memory.

ANRITSU CANNOT COMPENSATE FOR ANY MEMORY LOSS.

Please pay careful attention to the following points. Do not remove the IC card and backed-up memory from equipment being accessed.

(PMC)

- Isolate the card from static electricity.
- The back-up battery in the SRAM card has a limited life; replace the battery periodically.

(Backed-up memory)

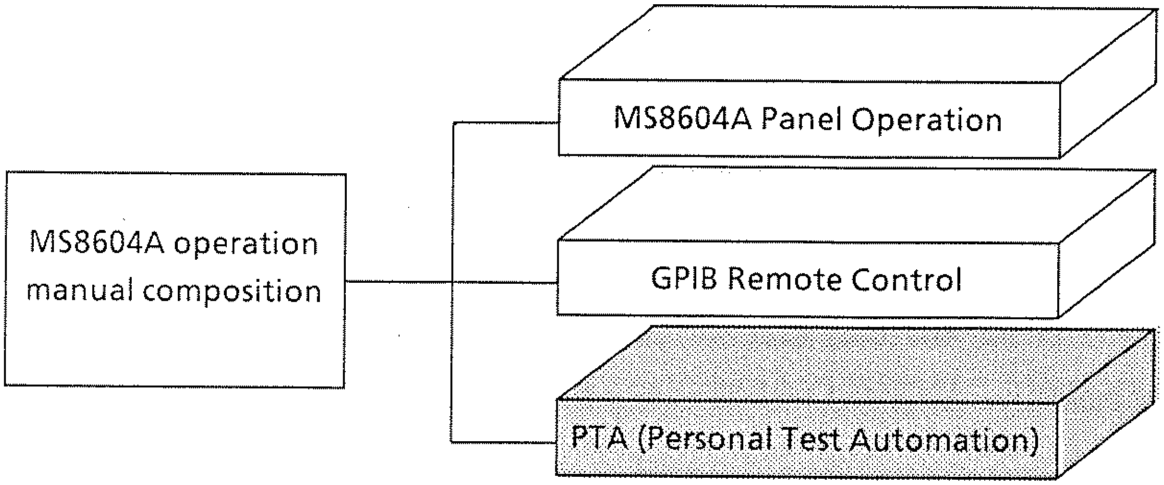
- Isolate the memory from static electricity.

Note: The battery life is about 7 years. Early battery replacement is recommended.

ABOUT THIS MANUAL

(1) Operation manual composition

The MS8604A Operation manuals are made up of the following three manuals. Use the manuals matching the usage objective.


MS8604A operation manual composition
MS8604A Panel Operation
GPIB Remote Control
PTA (Personal Test Automation)

Panel Operation:	Outlines the MS8604A and describes its specifications, preparations, panels, manual (local) operation method, storage, transportation, function keys transition diagram, initial values table, and error messages.
GPIB Remote Control:	Since the MS8604A is compatible with IEEE488.2, this manual describes GPIB remote control based on IEEE488.2. The descriptions in this manual are based on N_{88}-BASIC programs using an NEC PC9800 Series personal computer for program generation reference.
PTA Personal Test Automation:	Describes how to program high-speed control and high-speed operation processing directly connected to the measurement system by high level language PTL. The program is executed by a personal computer built into the MS8604A and is called PTA (Personal Test Automation). Together with GPIB mentioned above, it promotes measurement automation.

(2) Basic Guide to GPIB (sold separately)

A “Basic Guide to GPIB” is sold separately from the operation manuals above.
This guide provides a basic knowledge of GPIB and describes the GPIB control statements written in Anritsu PACKET V computer language.

TABLE OF CONTENTS

SECTION 1
GENERAL

TABLE OF CONTENTS

(Blank)

The Personal Test Automation (PTA), installed in the MS8604A Digital Mobile Radio Transmitter Tester, is a high-speed controller for system and calculation. The MS8604A with PTA can be used as a personal computer and programmed in the Personal test Language (PTL), a high-level programming language.

In addition to the basic commands similar to BASIC, PTL provides GPIB control commands, Plug-in Memory Card (PMC) control commands, I/O port control commands, screen control commands and function control commands for controlling most functions of the MS8604A.

Programs are entered, edited and executed by using the full keyboard and keys on the front panel of the MS8604A; programs made by using personal computer can also be transferred via the GPIB. In addition, since programs can be stored in the built-in, nonvolatile program memory and PMC, no efforts required for reloading programs are necessary. Further, by using the MC8104A Data Storage Unit, programs can be stored on floppy disk (FD), which is particularly effective when storing large programs.

The PTA uses the GPIB, RS-232C (option 02), or I/O port for external interfacing. The I/O port can easily be connected to an automatic inspection machine for electronic components or a timing machine, which can then be controlled at high speed using program.

In such a case, the GPIB is connected with an external computer to facilitate communication between the PTA and external computer through communication memory (dual port memory).

1.1 PTA Specifications

The PTA specifications are listed below:

■ Display

● Number of displayed characters :	80 characters (up to 68 characters can be input) × 25 lines
● Displayable characters :	Alphabetic upper- and lower-case characters, numerals, special symbols, and cursors
● Character font :	8 × 16 dots (small type)
● Graphics :	Straight line, square, circle and arc
● Screen :	640 × 400 dots × 4 screens (High brightness-1 screen, Intermediate brightness-2 screens, Low brightness-1 screen)

■ Input and execution control

● Input :	Full keyboard, front panel, and external computer (by GPIB)
● Execution control :	Full keyboard, front panel, and external computer (by GPIB)

■ Memory

● Program memory :	900 kbytes
● PMC :	32 kbytes, 128 kbytes, 256kbytes, 512kbytes
● Floppy disk (FD) :	Available only when the MC8104A Data Storage Unit is used

■ Language Version PTL - V1.5

● Commands :	Edit commands Program execution commands File commands
● Statements :	Basic statements GPIB statements Event statements I/O port statements Dual port statements
● Subroutines :	Display subroutines Plot subroutines Buzzer subroutines PMC subroutines Interface subroutines Panel subroutines Memory subroutines
● Functions :	Arithmetics functions Boolean functions Character string functions Statistical functions

● Variables :	Up to a maximum of 256 user-defined variables (numeric variable, character string variable) System variables 29 types
● Data types :	Real number: Significant digits = 15 digits Exponential = 10^{308} to 10^{-307} Integer −32768 to 32767 Character: 256 characters max. Bit: 8 bits max.

■ Interfaces :

First interface

● GPIB1 (IEEE-488) :	SH1,AH1,T6,L4,SR1,RL1,PP0,DC1,DT1,C1,C2,C3, C24

Second interface

● GPIB2 :	SH1,AH1,T6,L4,SR0,RL0,PP0,DC0,DT0,C1,C2,C3,C4,C28, E2

Third interface

- RS-232C
  (option 02)

● I/O ports:	Output port A	8 bits
	Output port B	8 bits
	Input/Output port C	4 bits
	Input/Output port D	4 bits
	Control port	3 bits

1.2 PTL Command of PTA

Table 1-1 shows the PTL (Personal Test Language) commands provided with the PTA :

Table 1-1 PTL Command of PTA

Item	Format
Edit Commands	
Program input	Line number statement
Auto	AUTO [start-line number] [, increment]
Copy	PCOPY new start-line number, [increment], copy-source start-line number, copy-source stop-line number
Delete	DELETE [start-line number][,[stop-line number]] or [line number] [RETURN]
Renumber	RENUM [new start-line number[,increment[, old start-line number[old stop-line number]]]]
List output (CRT)	LIST [start-line number] [,[stop-line number]
List output (printer)	LISTG address [,start-line number] [,[stop-line number]]
Program size	PMEMO
Page scroll	[PAGE SCROLL] key, [SHIFT]+[PAGE SCROLL] keys
Roll up/down	[CTRL]+[J] keys, [CTRL] + [K] keys
Execute commands	
Program execution start	[RUN] key or RUN [start -line number] [, suspension-line number]
Suspension of program execution	[STOP] key, [CTRL] + [C] keys
Continuation of suspended program execution	[CONT] key, CONT [suspension-line number]
Step execution	[STEP] key
Discontinuation of program execution	[RESET] key
Direct execution	Statement [RETURN]
File commands	
Save file	SAVE program name [, start-line number [, stop-line number]]
Load file	LOAD program name
Overlay	OVERLAY
File list display	[PLIST] key
Delete file	PDEL Program name
Start-up registration	STARTP program name or STARTP @
Start-up cancel	CANCEL or CANCEL @

Table 1-1 PTL Command of PTA (Continued)

Item	Format
Statements	
Comments	REM [“comment”] or ’comment
Array declaration	DIM array variable
Assignment	[LET] variable = expression (functions, variables or constants)
Jump	GOTO line number or GOTO * label
Jump to subroutines	GOSUB line number or GOSUB * label
Return from subroutines	RETURN
Decision	IF condition statement
Loop beginning	FOR numeric variable = initial value TO ending value STEP step value
Loop end	NEXT numeric variable
Key input	INPUT [“display character string” ,] variable [, variable...]
Display	PRINT variable [: format] [, variable [: format]...] [;]
Reverse display	PRINTR variable [: format] [, variable [: format]...] [;]
GPIB input	READ address, input variable [, variable...]
GPIB input (1 byte)	BREAD address, input variable [, variable...]
GPIB input (2 bytes)	WREAD address, input variable [, variable...]
GPIB output	WRITE address, variable [: format] [;]
GPIB output (1 byte)	BWRITE address, variable [: format]
GPIB output (2 bytes)	WWRITE address, variable [: format]
Set measurement parameter	PUT string variable (or string)
Read measurement parameter (1)	GET string variable (or string), input variable
Read measurement parameter (2)	COM character string variable (or character constant) > input variable
Wait	WAIT time (unit is second, minimum 0.01 s.)
Subroutine call	CALL subroutine name
Cursor location (home position)	HOME
Cursor location	LOCATE (X, Y)
Erase screen	ERASE
Program end	STOP
Display error	Line NO _ SOS_ “Grammer error expression“
Jump on Error	ERROR (error number, line number or * label)
Error main	ERRMAIN
Return to main routine	RETMAIN
Initialization of variable	CLEAR

Table 1-1 PTL Command of the PTA (Continued)

Item	Format
Statement (cont'd)	
Data statement	DATA constant [, constant..., constant]
Specification of input data statement	RESTORE [line number or * label]
Data-statement input	RDATA variable [, variable....]
Program reading/execution	CHAIN "file name"
Register an error interrupt routine	ON ERROR line number or * label
Unregister an error interrupt routine	OFF ERROR
Return from an error interrupt routine	RETERR RETRY RESUME line number or * label GIVEUP
Register an event interrupt routine	ON EVENT I/O number, line number or * label
Enable an event interruption	ENABLE EVENT I/O number, event 3, event 2, event 1, event 0
Disable an event interruption	DISABLE EVENT I/O number [, event 3, event 2, event 1, event 0]
Return from an event interrupt routine	RETINT
Jump on interrupt (I/O port)	ON IO GOTO line number or * label
Jump on interrupt subroutine (I/O port)	ON IO GOSUB line number or * label
Enable interruption (I/O port)	IOEN
Disable interruption (I/O port)	IODI
Mask interruption (I/O port)	IOMA
Character size specification	DCHSIZE character size number
Write strobe signal switch (I/O port)	OLDPORT
Pseudorandom number string setting	RNDMIZE
Dual-port-memory statement	
Write data	WDPM memory No., variable [: format]
Read data	RDPM memory No., input variable
Screen subroutines	
Displayed-item erasure	CALL CER (M)
Displayed-item restoration	CALL CRN (M)
Screen erasure	CALL CFL (M)
Character string display	CALL DCH (X, Y, text, M[, N])

Table 1-1 PTL Command of PTA (Continued)

Item	Format
Screen subroutines (cont'd)	
Straight-line display	CALL DLN (X0, Y0, X1, Y1, M[, N])
Square display	CALL DRC (X0, Y0, X1, Y1, M[, N])
Circle display	CALL DCR (X, Y, R, M[, N])
Arc display	CALL DAR (X0, Y0, R0, W1, W2, M[, N])
Soft key label registration	CALL DEF (M, text)
Buzzer subroutine	
Buzzer	CALL BZR
PMC subroutines	
Open a file (read)	CALL OPNI Character string variable (or character constant)
Open a file (write)	CALL OPNO Character string variable (or character constant)
Delete a file	CALL FDEL Character string variable (or character constant)
Load data	CALL DALD variable
Save data	CALL DASV variable
Close a file	CALL CLS
Save function	CALL FNSV (M)
Recall function	CALL FNRC (M)
Panel subroutines	
Lock front-panel key operation	CALL PNLL (0)
Unlock front-panel key operation	CALL PNLU (0)
Waveform memory subroutine	
Copy memory	CALL COPY (M0, M1)
Video plotter control subroutine	
Start copy	CALL VPT
GPIB subroutine (for GPIB1 port only)	
Interface clear (switching to system controller port)	CALL IFC
Service request	CALL RSV(M)
Take controller	CALL TCT(M)
Switching to device port	CALL DEV
Interface subroutine	
Status byte read	CALL GST (port No., address, input variable)
Interface control	CALL GPIB (port No., control item No.)

Table 1-1 PTL Command of PTA (Continued)

Item	Format
Function	
Arithmetic functions	SIN, COS, TAN, ASN, ACS, ATN, LN, LOG, EXP, SQR, ABS, SGN, INT, ROUND, DIV, FIX
Boolean functions	NOT, AND, OR, EOR
Character string functions	CHR, VAL, HVAL, BVAL, ASC, CHR$, CVI, CVD, MKI$, MKD$, STR$, HEX$, OCT$, BIN$, INSTR, LEFT$, MID$, RIGHT$, STRING$, LEN, SLEN, SGET$
Statistical functions	max, min, sum, mean, var, sta
Dedicated functions	ERRREAD, STATUS, DTREAD$, RND
System variable	EX0, EX1, EX2, EX3, EX4, EX5, EX6, IOA, IOB, IOC, IOD, EIO, DT0, DT1, DT2, DT3, DT4, XMA, XMB, XMC, XMD, XME, XMT, SMA, SMB, IMA, IMB, RMA, RMB

Table 1-2 I/O Port Connector (RC30-36R : Hirose Electric, Japan)

PIN No.	Item	Specification	System variable name
1	GND		
2	INPUT 1	TTL level, negative logic	EXØ
3	OUTPUT 1		
4	OUTPUT 2		
5	OUTPUT PORT A 0	TTL level, negative logic	IOA
6	OUTPUT PORT A 1		
7	OUTPUT PORT A 2		
8	OUTPUT PORT A 3		
9	OUTPUT PORT A 4		
10	OUTPUT PORT A 5		
11	OUTPUT PORT A 6		
12	OUTPUT PORT A 7		
13	OUTPUT PORT B 0	TTL level, negative logic	IOB
14	OUTPUT PORT B 1		
15	OUTPUT PORT B 2		
16	OUTPUT PORT B 3		
17	OUTPUT PORT B 4		
18	OUTPUT PORT B 5		
19	OUTPUT PORT B 6		
20	OUTPUT PORT B 7		

Table 1-2 I/O Port Connector (Continued)

PIN No.	Item	Specification	System variable name
21	INPUT/OUTPUT PORT C0	TTL level, negative logic	IOC
22	INPUT/OUTPUT PORT C1		
23	INPUT/OUTPUT PORT C2		
24	INPUT/OUTPUT PORT C3		
25	INPUT/OUTPUT PORT D0	TTL level, negative logic	IOD
26	INPUT/OUTPUT PORT D1		
27	INPUT/OUTPUT PORT D2		
28	INPUT/OUTPUT PORT D3		
29	PORT C STATUS	TTL level, 0/1 : Input / Output	EIO
30	PORT D STATUS	TTL level, 0/1 : Input / Output	
31	Write strobe signal	TTL level, negative logic	*1
32	Interrupt signal	TTL level, negative logic	*2
33	(NC)		
34	+5V Output	Max. 100mA	
35	(NC)		
36	(NC)		

*1 This is output at IOC=..., IOD=... statement execution.

*2 The interrupt is executed by using such I/O interrupt statements as ON IO GOTO... and ON IO GOSUB....

Notes: The I/O port is available as standard.

See SECTION 9 for details on the I/O PORT CONTROL.

The name of the above connector is RC30-36R (Hirose Electric, Japan). As the connector for the RC30-36R is RC30-36P (Hirose Electric, Japan), prepare it as required.

1.3 MS8604A External Interface

The MS8604A is equipped with two GPIB interface ports as standard.

The GPIB1 is mostly used for controlling the MS8604A from an external controller which is expected in general GPIB we call this as device mode. Contrarily, the GPIB2 is used for controlling other devices by the MS8604A without the external controller.

The following external interfaces is also available for the MS8604A:

- GPIB1 port (interface 1: Standard)
- GPIB2 port (interface 2: Standard)
- RS-232C port (interface 3: Option 02)
- Parallel I/O port (Standard)

1.3.1 GPIB1 port

When the PTA is OFF, this port acts as a device port and is used to control the MS8604A from an external computer.
When the PTA is ON, this port operates as a system controller port.
However, when the PTA is turned off, this port reverts to device mode.
The device/controller mode can be switched by PTA statement.

The following devices can be connected to this port:

- Device port mode
  Host computers and any other controllers which control the MS8604A.
- System controller mode
  Slave devices (devices controlled from the MS8604A).

Notes:
- The MC8104A (Data Storage Unit) cannot be connected to this port.
- Talk only and listen only modes cannot be used.
- Do not connect GPIB1 with GPIB2.

1.3.2 GPIB2 port

This port acts as a system controller port nevertheless the PTA is on or off. Therefore, it is used to control slave devices from the MS8604A.

The following devices can be connect to this port:

Slave devices (devices controlled from the MS8604A)
Printer, MC8104A (Data Storage Unit)

Notes:
- Talk only and listen only modes cannot be used.
- Do not connect GPIB1 with GPIB2.

1.3.3 RS-232C port

This port is used only when the RS-232C (Option 02) interface is installed.
This port can be used for serial data transfer.
MODEM control, etc. can also be programmed by the PTA.

The following devices can be connected to this port:

MODEM
Other devices with an RS-232C interface

Notes:
- The MC8104A (Data Storage Unit) cannot be connected to this port.
- The MS8604A measurement parameters cannot be controlled directly by connecting a host computer to this port. In this case, the MS8604A must be controlled via the PTA.

1.3.4 Parallel I/O port

This port controls devices without GPIB and RS-232C interfaces and devices without a special data transfer protocol and handshake.
External devices can be easily controlled by the PTA statement.

The following devices can be connected to this port:

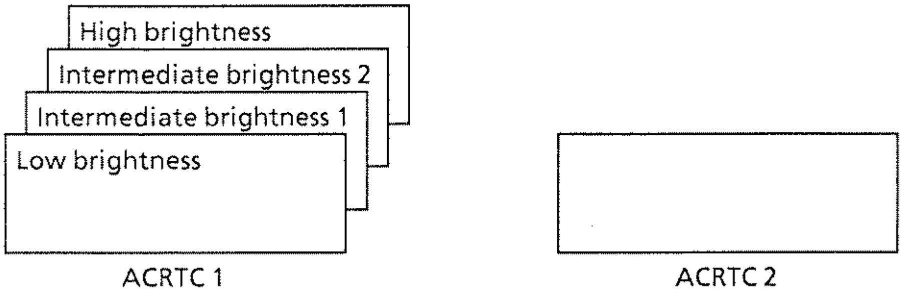
Devices without a GPIB and RS-232C interfaces
(Control of handler, switch, lamp, relay, etc.)

Note: The MC8104A (Data Storage Unit) cannot connect to this port.

1.4 Screen Configuration of PTA

This section describes the screen specifications of PTA mounted in the MS8604A.

1.4.1 Physical screen configuration



(1) ACRTC1 (for character display)

Low brightness screen : Scale lines, scale coordinates, menu frames, display parameters, error scale lines

Intermediate brightness 1 screen : PTA screen

Intermediate brightness 2 screen : Title characters, Communication Message Area (CMA) characters, menu characters, setup and display parameter characters, error messages, cursor, measurement results

High brightness screen : Markers, zones indicating occupied bandwidths

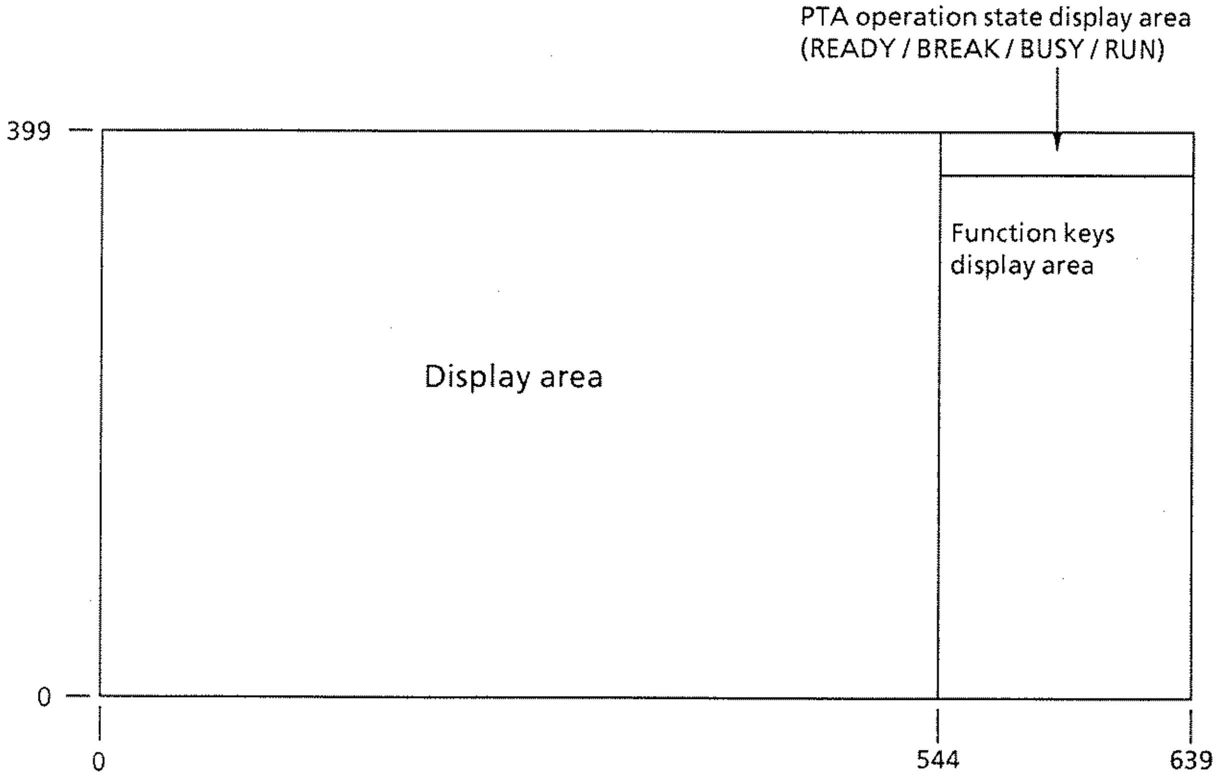
(2) ACRTC2 (for waveform display)

Constellation waveform, power waveform, occupied bandwidth waveform, adjacent channel leakage power waveform, spectrum waveform

Template

Line being selected at template modification

1.4.2 Display form


PTA operation state display area
(READY / BREAK / BUSY / RUN)
399
Function keys
display area
Display area
0
0
544
639

Notes:
- When a PTA function key is pressed, execution starts immediately. Accordingly, double frame display and key label reversing are not performed.
- " * " is displayed at the upper right of a key label for keys with menu development in the MS8604A. However, " * " is not displayed at the [etc : F6] key of each level for the PTA function keys.

SECTION 2
OPERATION

TABLE OF CONTENTS

(Blank)

2.1 Operation Outline

The PTA inputs, edits, and executes the program via the external keyboard. Also, the PTA can execute the program using front-panel keys and can input and execute the external computer program via GPIB1.

2.2 PTA Activation

Press the [PTA] key on the front panel of the MS8604A or input PTA 1 from the GPIB1 to turn on the PTA. At this time, the CRT screen is cleared and the cursor is displayed at the HOME position (upper-left corner of the screen). The following menus corresponding the soft keys [F1] through [F6] are displayed.

When the PTA program is startup-registered, PTA is activated at power on, and the PTA program can be executed.
(See Paragraph 3.22, " STARTP Command " for details of startup-registration of the PTA program.)

RUN	F1	....	Starts program execution.
STOP	F2	....	Suspends program execution.
CONT	F3	....	Restarts suspended-program execution.
RESET	F4	....	Stops program execution.
PTA OFF	F5	....	Releases PTA.
etc(1/4)	F6	....	Selects the next PTA menu.

Notes:

- The program in the PTA memory is not lost if [PTA OFF : F5] or the power switch of the MS8604A is pressed. Afterwards, if the PTA is set to ON by pressing the [PTA] key, the program is automatically loaded and executed.
- Since program is not lost by pressing PTA OFF, when a new program is loaded after PTA ON, the previous program is erased and then the new program is loaded.
- When overwrite execution is required at loading with the previous program not erased, load the new program after executing the OVERLAY command.

2.3 Panel Operation in PTA

2.3.1 Soft keys

Each time the [etc : F6] soft key is pressed when the PTA is ON, the soft key menus are displayed as shown below:

⇓

PLIST	F1		……	Causes the names of the PTA file stored in the PMC to be listed on the CRT.
CURSOR UP	F2		……	Moves cursor up
CURSOR DOWN	F3		……	Moves cursor down
LOAD	F4		……	Loads program indicated by file name displayed on cursor line
RUN	F5		……	Starts program execution
etc(2/4)	F6		……	Selects the next menu

⇓

F1	F1	(*)	……	Function key
F2	F2	(*)	……	Function key
F3	F3	(*)	……	Function key
F4	F4	(*)	……	Function key
F5	F5	(*)	……	Function key
etc(3/4)	F6		……	Selects the next menu

(*) The displayed characters can be defined by the DEF subroutine.

⇓

(To the next page)

(From the previous page)

YES	F1	......	Inputs "YES" character string
NO	F2	......	Inputs "NO" character string
(None)	F3	......	None
(None)	F4	......	None
(None)	F5	......	None
etc(4/4)	F6	......	Returns to the first PTA menu.

2.3.2 Inputting, executing, and stopping program

The program is created by inputting and editing via the external keyboard, and can be saved in the PMC (Plug-in Memory Card). Execute the following operation to load and execute the program saved in the PMC via the front panel (The program can not be saved to the PMC from the front panel key.).

① Press the [PLIST : F1] key of the PTA menu (2/4) to display the program names (stored in the PMC) on the CRT

② Move the cursor to the line of the program name to be loaded with the [CURSOR UP : F2] and [CURSOR DOWN : F3] keys.

③ Press the [LOAD : F4] key to load the program.

④ Press the [RUN : F5] key to start execution.

⋮

⑤ Press the [RESET : F4] key of the PTA menu (1/4) to stop execution.

2.3.3 Data input keys

The soft keys, numeric keys, and unit keys on the front panel serve as data input keys.

(1) F1, F2, F3, F4 and F5 keys

The F1 to F5 keys are referred to in the program and correspond to the system variables EX1, EX2, EX3, EX4 and EX5 respectively.
Each time the key is pressed, the variable contents are alternately changed to 0 or 1. All the data in these variables are 0 at initial state and resetting. Displayed name in menu can be defined with DEF subroutine.

Note: For EX1, EX2, EX3, EX4 and EX5, refer to paragraph 5.1.

(2) YES and NO keys

These are typing aids for the INPUT statement; the"YES" and "NO" character string can be input by a single key operation.

(3) Numeric keys

These are the [0] to [9] ,[.] and [BS] keys which are used for inputting data on INPUT statement.

Press the [Enter] key to terminate the input; use the [BS] key to delete one character.

(4) Unit keys

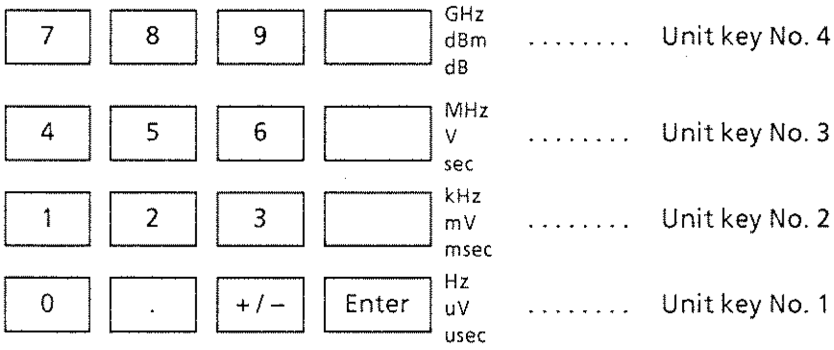
Unit key No. 1 : Treats this key as the CR key.

Unit key No. 2 : Treats this key as the [,] key.

Unit key No. 3 : Treats this key as the [–] key.

Unit key No. 4 : Invalid

* : The figure below shows unit key numbers.



2.3.4 Operation of other panel keys

When PTA is ON, the panel keys are locked-out except for the number / [Enter] keys, [Shift] key, [Local] key and soft keys (F1 to F6).

2.4 PTA Termination

Press the [RESET : F4] to stop program execution and then [PTA OFF : F5] or input PTA 0 from the GPIB1 to terminate PTA operations.
Afterwards, the screen (which has been displayed by display subroutine) is cleared to be returned to ordinary measurement screen.

Note: For the display subroutine, refer to paragraph 5.2.

2.5 External Keyboard

Input, edition and execution of the program can be operated via the external keyboard.
When PTA is turned on, data entry from the PTA external keyboard usually becomes valid. However, even if PTA is off, title entry becomes valid.

2.5.1 External keyboard connection

Connect the external keyboard to the [Keyboard] 8 pins DIN connector on the MS8604A front panel.

CAUTION

Connect and disconnect the external keyboard with the MS8604A power switch in the OFF position.

2.5.2 Explanation of external keyboard

Figure 2-1 shows the external keyboard.

(1) Character, numeric and symbol keys

These keys input the command, program, and data.

(2) [SHIFT] key

Press a character, numeric, or symbol key while pressing the [SHIFT] key to input the upper-case alphabetic character(lower-case alphabetic character when [CAP LOCK] is ON) or the upper symbol on the key.

(3) [CAP LOCK] key

When this key is ON, a lamp lights and the alphabetic characters becomes upper case.
When this key is OFF, the alphabetic characters becomes lower case. Use [CAP LOCK] key when PTA is turned ON.

(4) [↵] key ([RETURN] key)

This key terminates command, one-line program, or one-line data input.

(5) [TAB] key

This key inputs a fixed-length space.

(6) [HOME ERASE] key

This key erases the CRT screen and displays the cursor at the HOME position (upper-left corner of the screen).

(7) [LIST] key

This key displays the "LIST" character string of the list command on the CRT.

(8) [RENUM] key

This key displays the "RENUM" character string of the renumber command (for program line renumbering) on the CRT.

(9) [DLT] key

This key displays the "DELETE" character string of the delete command (program deletion) on the CRT.

(10) [INS] key

This key changes the mark of cursor to ◀ (insertion cursor). Press the character, numeric, and symbol keys to insert them in front of the insertion cursor. If you press this key again, the cursor returns to the normal cursor.

(11) [DEL] key

This key deletes the character at the cursor.

(12) [BS] key

This key deletes the character before the cursor.

(13) ↑, ↓, → and, ← keys

These keys move the cursor up or down, and right or left. When the ↑, ↓, → or ← key is pressed with the [SHIFT] key, the cursor moves the end of upper right or left or lower right or left on the screen.

(14) [F1], [F2], [F3], [F4] and [F5] keys

These keys can be referred to in the program and correspond to the system variables, EX1, EX2, EX3, EX4 and EX5, respectively. Each time the key is pressed, the variable contents are alternately switched to 0 or 1.

(15) [RUN] key

This key starts program execution from the first line.

(16) [SAVE] key

This key displays the "SAVE" character string of the save command on the CRT.

(17) [LOAD] key

This key displays the "LOAD" character string of the load command on the CRT.

(18) [PLIST] key

This key displays the file names and file sizes stored in PMC, and the unused memory sizes of PMC, on the CRT.

(19) [RESET] key

This key stops program execution.
The each variable contents and the I/O port are initialized.

(20) [STOP] key

This key suspends program execution.
The variable contents are not initialized

(21) [CONT] key

This key restarts program execution after it has been suspended.

(22) [STEP] key

This key executes the program in one-line steps.

(23) [PAGE SCROLL] key

This key displays the program on the CRT in one-page units (24 lines).

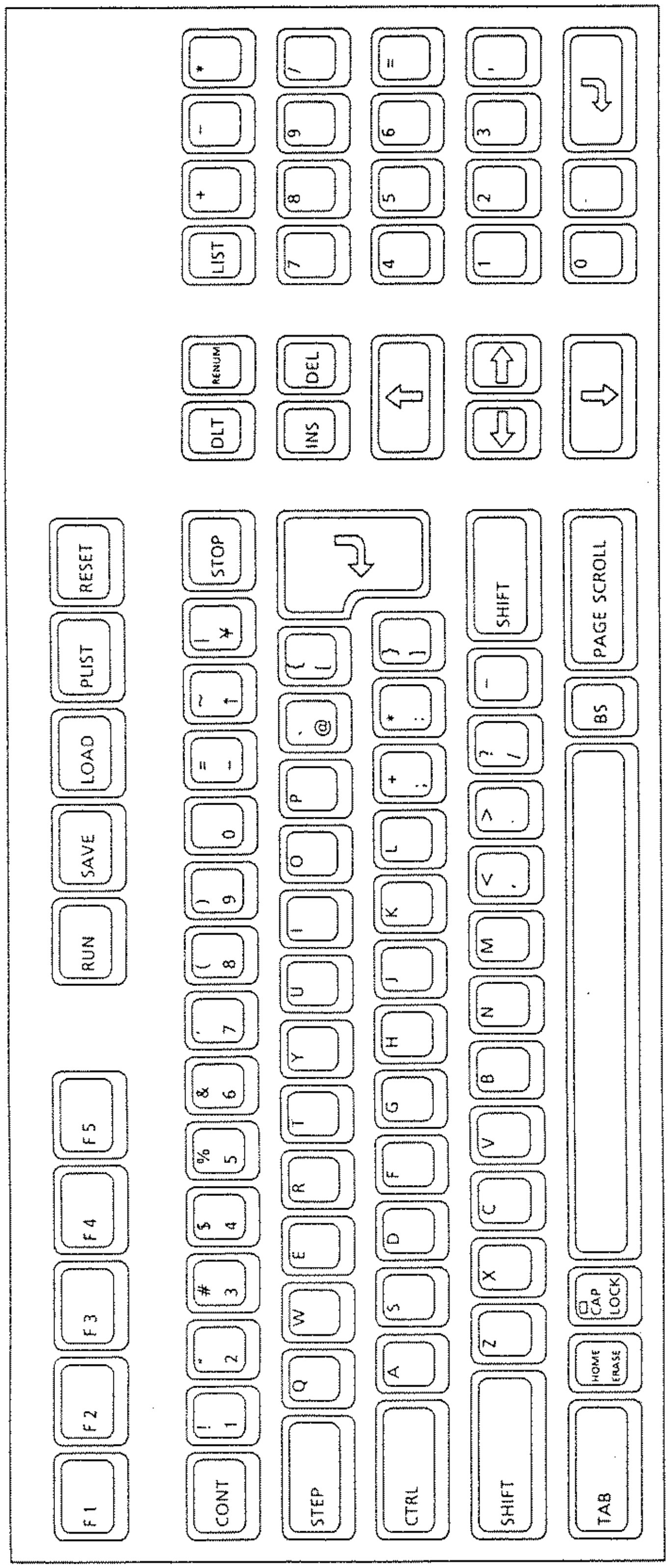


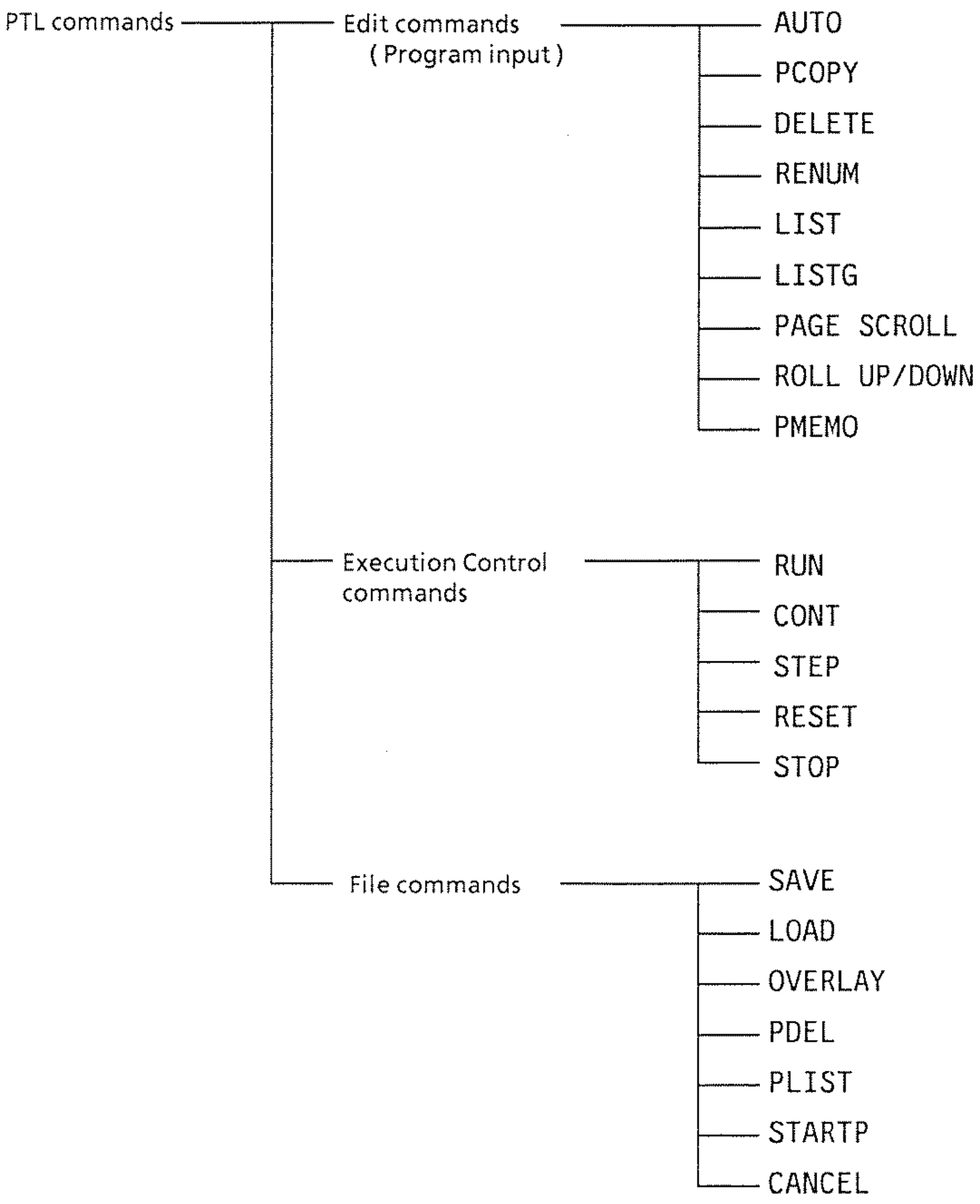
Fig. 2-1 PTA External Keyboard

SECTION 3
PTL COMMANDS

TABLE OF CONTENTS

(Blank)

Programs written in PTL (Personal Test Language) are edited, executed, and filed by PTL commands. The PTL commands and their functions are as follows :



- PTL commands
  - Edit commands (Program input)
    - AUTO
    - PCOPY
    - DELETE
    - RENUM
    - LIST
    - LISTG
    - PAGE SCROLL
    - ROLL UP/DOWN
    - PMEMO
  - Execution Control commands
    - RUN
    - CONT
    - STEP
    - RESET
    - STOP
  - File commands
    - SAVE
    - LOAD
    - OVERLAY
    - PDEL
    - PLIST
    - STARTP
    - CANCEL

3.1 Program Input Command

(1) Function

When a statement with a line number is input, this command stores the statement in the program area as a program.
If the line number is different from the line numbers of previously stored statements, the statement is added to or inserted between the previously stored program.
If the line number is the same as that of a stored statement, the previously stored statement is overwritten by the new statement.

(2) Format

```
Line number    Statement
  |
Integer constant from 1 to 65535
```

Notes:

- When 111 or more characters (including the line number) are input on one line during program input, the program on that line may not be displayed during LIST-command execution after execution of the RENUM command.
- For a description of the RENUM command, see paragraph 3.5.

3.2 AUTO Input Command

(1) Function

This command automatically displays the next line number for the program.
It causes the <start-line number> to be displayed and awaits for statement input.
When a statement is entered and the return key pressed, the statement is stored in the program area.
In addition, the line number is updated by the specified <increment>.
If the <start-line number> or <increment> is omitted, '10' is used as the default value.
Press the [RESET] key to cancel the AUTO input command.

(2) Format

```
AUTO [operand 1] [operand 2]
         |              |
     Start number    Increment
```

① AUTO Updates the line number in increments of 10 from line 10.

② AUTO 1ØØ Updates the line number in increments of 10 from line 100.

③ AUTO ,1 Updates the line number in increments of 1 from line 10.

④ AUTO 1ØØ,5 Updates the line number in increments of 5 from line 100.

Note: Line numbers are incremented in accordance with the specified <increment> regardless of whether or not the new line number overwrites a previously used line number.

3.3 PCOPY Command

(1) Function

This command copies the specified program.
(from <copy-source start-line number> to the <copy-source end-line number>) in the unit of incrementation specified by <increment> from the <new start-line number>.
If <increment> is omitted, then '10' is used as the default value.

(2) Format

```
PCOPY operand 1,[operand 2],operand 3,operand 4
         /            |              |            \
New start-line number  Increment  Copy-source start-line number  Copy-source end-line number.
```

① `PCOPY 100,,10,30` Copies the program (from lines 10 to 30) to location 100 in increments of 10 and labels all sequent.

② `PCOPY 100,5,10,30` Copies the program (from lines 10 to 30) to location 100 and in increment of 5 and labels all sequent.

Notes:

- If the line number of a newly-copied program is identical to the line number of the current program, ERROR F101 occurs.
- If a line has more than 111 characters when PCOPY is executed, display is disabled during LIST command execution.

3.4 DELETE Command

(1) Function

This command deletes all or part of a program.

(2) Format

```
DELETE [operand 1] [,] [operand 2]
           |                |
   Start line number     End line number
      operand 1≦operand 2
```

(3) Example

① `DELETE` Deletes entire program and initializes variable values.

② `DELETE 100` Deletes statement on line 100.

③ `DELETE 100,` Deletes statements on lines 100 to the end line.

④ `DELETE ,500` Deletes statements on start line to line 500.

⑤ `DELETE 100,500` Deletes statements on line 100 to line 500.

- When deleting only a line, it is possible by Line number [RETURN].

3.5 RENUM Command

(1) Function

This command renumbers line numbers used in the program. When the increment value or new line number is omitted, 10 is used as the default value.

(2) Format

```
RENUM [[operand 1] [,] [operand 2] [,] [operand 3] [,] [operand 4]]
            |                 |                 |                 |
     New-line number      Increment      Start-line number   End-line number
```

① `RENUM` Renumbers all program statements starting from first-line number 10 in increments of 10

② `RENUM 100` Renumbers all program statements from new first-line number 100 in increments of 10

③ `RENUM 100,5` Renumbers all program statements starting from new first-line number 100 in increments of 5

④ `RENUM 100,,50` Renumbers statements (from line number 50 through the last line) starting from new first-line number 100, in increments of 10

⑤ `RENUM 100,5,50` Renumbers statements (from line number 50 through the last line) starting from new first-line number 100, in increments of 5

⑥ `RENUM 100,,,150` Renumbers statements (from line number 10 through 150) starting from new first-line number 100 in increments of 10

⑦ `RENUM 100,,50,150` Renumbers statements (from line number 50 through 150) starting from new first-line number 100 in increments of 10

⑧ `RENUM 100,5,,150` Renumbers statements (from line number 10 through 150) starting from new first-line number 100 in increments of 5

⑨ `RENUM 100,5,50,150` Renumbers statements (from line number 50 through 150) starting from new first-line number 100 in increments of 5

Notes:

- Labels can be used for operands 1, 3 and 4.
- "ERROR F101" occurs if there is a line number larger than that of operand 4 when operand 1 is smaller than operand 4.
- If the number of characters on a line is more than 111 characters, when the number of lines of the program line becomes two lines or more with RENUM command, ERROR F20 will occur during LIST command execution and display the lines.

3.6 LIST Command

(1) Function

This command displays all or part of a program on the CRT screen.

(2) Format

```
LIST [operand 1] [,] [operand 2]]
         |                |
  Start-line number    End-line number
        operand 1≦operand 2
```

①	LIST	Lists entire program
②	LIST 100	Lists the statement on line 100
③	LIST 100,	Lists statements on line 100 to end line
④	LIST ,500	Lists statements on start line to line 500
⑤	LIST 100,500	Lists statements on line 100 to 500

Note: Labels can be used for operands 1 and 2.

3.7 LISTG Command

(1) Function

This command outputs all or part of a program to the printer connected to the GPIB.

(2) Format

```
LISTG address [[,][operand 1][,][operand 2]]
      |                 |                |
Address of printer   Start-line number   End-line number
(0 to 30)
```

Operand 1 and operand 2 in the LISTG command are used in the same way as the LIST command.

Note:
- When printing via the GPIB1 port (the first interface port), set the GPIB1 port to system controller by issuing the CALL IFC statement. Note that the GPIB1 port is initially a control port in its in itial state. See paragraph 6.3 for details on switching the control port.
- When the contents of the program are output to the RS-232C, addresses have no meaning. However, addresses should still be specified for form's sake.

3.8 PAGE SCROLL Command

(1) Function

This command outputs the program in one -page (for 24 lines) units as follows:

First, output the program on the CRT screen using the LIST command. Program listing stops at the first PAGE SCROLL command. Then, output it to the last line in one page units by the following PAGE SCROLL commands.

(2) Format

```
[PAGE SCROLL] key
[SHIFT] key + [PAGE SCROLL] key
```

Notes:

- The [SHIFT] key + [PAGE SCROLL] key command scrolls lines backward.
- If the program is not displayed on the CRT screen via the LIST command, then the PAGE SCROLL command is not executed.

3.9 ROLL UP/DOWN Command

(1) Function

This command scrolls lines forward or backward when at least one program line is listed on the CRT screen.

(2) Format

```
[CTRL] key + [J] key → ROLL UP
[CTRL] key + [K] key → ROLL DOWN
```

Note: When the AUTO command is executed after executing the RUN command or after entering [HOME ERASE], and the screen is cleared; subsequent execution of the ROLL UP/DOWN command is disabled.

3.10 PMEMO Command

(1) Function

This command displays the used memory size of the program area in which a program is stored, and the file size required to store the same program in the PMC or FD .

(2) Format

```
PMEMO
```

(3) Output example

program size : 123 bytes	Used memory size of the program area
file size : 512 bytes	Required file size for storing in the PMC or FD.

3.11 Immediate Execution Command

(1) Function

When a statement with no line number is input and the ↵ (RETURN) key is pressed, the statement is immediately executed.

However, GOTO, GOSUB, RETURN, RETMAIN, IF, FOR, NEXT DATA, RDATA, RESTORE and CHAIN statements are not immediate execution commands.
See Section 4 for these statements.

(2) Format

```
Statement
```

3.12 RUN Command

(1) Function

This command starts program execution. When the STOP statement is executed, program execution is terminated. When an error occurs or the [RESET] key is pressed, program execution is stopped.

(2) Format

```
[RUN] key or
RUN [[operand 1][, operand 2]]
          |            |
Start-line number    Suspended-line number
```

①	RUN	Starts execution from statement on first line
②	RUN 1ØØ	Starts execution from statement on line 100
③	RUN ,5ØØ	Starts execution from statement on first line, and suspends execution on line 500
④	RUN 1ØØ,5ØØ	Starts execution from statement on line 100, and suspends execution on line 500

Note: Contents of variables are not initialized by the RUN command.

3.13 STOP Command

(1) Function

This command stops the program being executed.

(2) Format

```
[STOP] key
[CTRL] key + [C] key
```

3.14 CONT Command

(1) Function

This command restarts suspended program execution .

Note that this command can only be executed when program execution is suspended after execution of the RUN or STEP command.

(2) Format

```
[CONT] key
CONT [operand]
```

① CONT — Restarts program from statement on suspended line.

② CONT 1ØØØ — Restarts program from statement on suspended line, and suspends execution on line 1000.

3.15 STEP Command

(1) Function

This command executes only one statement line and displays the executed line number on the CRT screen.

When program execution is in termination status, the first line statement is executed at first. When it is in suspended status, the statement on the line next to the suspended line is executed.

(2) Format

```
[STEP] key
```

3.16 RESET Command

(1) Function

This command stops command or program execution.

(2) Format

```
[RESET] key
```

(3) Initialization

This Command : 1. Clears system variables EX1, EX2, EX3, EX4, and EX5.

2. Reinitializes the I/O port.

3. Clears user-defined variables.

3.17 SAVE Command

(1) Function

This command saves a program on the PMC or FD.

When it is used, the program file size must be smaller than the unused contiguous size of the memory space of the PMC or FD.

The file size and unused contiguous size are displayed on the CRT screen by executing the PMEMO and PLIST commands, respectively.

(2) Format

```
SAVE Program name [,operand 1] [,operand 2]
         |              |              |
         |       Start-line number   End-line number
         |
         └─Alphanumeric string up to 6
           characters starting with an
           upper-case alphabetic character.
```

Notes:

- The file opened by CALL OPNI (or OPNO) " % file name " is closed when this command is executed.
- Labels can be used as operands 1 and 2.
- The PMC is not formatted at the factory. When a program is saved into the PMC, format the PMC in advance.
  For formatting method of the PMC, refer to paragraph 4.5.2 of Panel Operation Part in the Operation Manual.

3.18 LOAD Command

(1) Function

This command loads the specified program stored in the PMC or FD into the user program area in the main frame.

All the programs already residing in the user program area are replaced by the new program unless OVERLAY is not executed.

(2) Format

```
LOAD program name
```

Notes:

- The file opened by CALL OPNI(or OPNO) "% file name" is closed when this command is executed.
- When reset during program loading, part of the programs is loaded.
- The MS8604A user program area (memory) is backed up by a battery. Therefore, the program contents are not lost even when the power switch is turned off.

3.19 OVERLAY Command

(1) Function

This command specifies to overwrite the current program during LOAD command execution.

(2) Format

```
OVERLAY
```

Note: This state continues until the RESET command is executed.

3.20 PDEL Command

(1) Function

This command deletes the programs stored in the PMC or FD.

(2) Format

```
PDEL program name
```

Notes:

- "% file name " (data files) cannot be erased by the PDEL command.
- The file opened by CALL OPNI (or OPNO) "% file name" is closed when this command is executed.

3.21 PLIST Command

(1) Function

This command displays on the CRT screen the names and sizes of files stored on PMC or FD along with the amount of unused memory.

(2) Format

```
[PLIST] key
```

(3) Output

This command causes the screen to scroll by page (24 lines) unit.
When more than 22 files are stored on a memory card, the files cannot be displayed on one page, therefore a screen such as ① below is displayed. The screen is displayed page by page by using the PLIST command repeatedly. When the contents can be displayed on a single page, a screen such as ② is displayed.

① When pages follow

```
.............              ..... bytes
%SDAT0              1024  bytes
%SDAT2              1024  bytes
ABCXYZ             15808  bytes
                                          continue
```

② When no pages follow

```
BANDLH             18568  bytes
RPLLH              35786  bytes
MAXMIN             27368  bytes
unused  memory  size :  89010  bytes
```

Unused memory size: Indicates unused memory size (No. of byte) of the PMC or FD.

File attribute:

- A file name (starting with an uppercase alphabetic character) indicates a program file.
- A file name (starting with %) indicates a data file.

Notes:

- The file opened by CALL OPNI (or OPNO) "% file name" is closed when this command is executed.
- Only the program file and data file created by the PTA are displayed by the PLIST command. Therefore, since the MS8604A does not display the saved waveform and measurement parameters, if they exist, the unused memory size is reduced.

3.22 STARTP Command

(1) Function

Turns on the PTA and registers the start-up function, which loads and executes the specified program when the power is turned on.
This function can be registered for PMC or FD; and can be registered and set individually for the MS8604A internal PTA program.

(2) Format

```
STARTP program name  : Register for PMC or FD
STARTP @             : Register for MS8604A internal program
```

① Start-up function registration for PMC or FD

- When the power is turned on after this function is registered, the PTA is turned on and the registered program is loaded and executed.
- When this function is registered, a special "ms8604. bat" file is created on the PMC or FD. (This file is not displayed by the PLIST command.)
- In the following cases, the start-up function is not performed even if registered:
  - When a PMC or FD is not inserted when the power is turned on.
  - When a PMC or FD is not inserted correctly at the selected media (INT PMC, EXT PMC, etc.)
  - When a program with the registered program name is not found on the PMC or FD.
  - If the power was turned on while pressing the [PTA] key.
- This function is executed first even if the start-up function is performed for the MS8604A internal program.
- When the start-up function is executed, the program is loaded from the PMC or FD and the previous program in the MS8604A is cleared. When the start-up function was performed for the program in the MS8604A, it is also cleared.
- If both "STARTP" and "STARTP@" are registered, the file registered by the STARTP command is executed preferentially.

② Start-up function registration for MS8604A internal program

- When the power is turned on after this function is registered, the PTA is turned on and the MS8604A battery back-up PTA program is run automatically.
- When there is no program in the MS8604A, this function cannot be registered.
- The start-up function is not performed in the following cases:
  - When the PMC or FD start-up function was executed first.
  - When a new program was loaded after the start-up function was registered. (In this case, start-up function registration is canceled.)
  - When there is no program in the MS8604A.
  - If the power was turned on while pressing the [PTA] key.

3.23 CANCEL Command

(1) Function

Cancels start-up function registration.

(2) Format

```
CANCEL      : Cancel registration for PMC or FD
CANCEL @    : Cancel registration for MS8604A internal program
```

- When start-up registration for PMC or FD is canceled, the "ms8604. bat" file is deleted.
- When the power is turned on while pressing the [PTA] key, the start-up function is temporarily canceled, but the function registration status does not change.

SECTION 4
PTL

TABLE OF CONTENTS

(Blank)

PTL (Personal Test Language) is a programming language similar to BASIC.

It consists of basic PTL statements and extended PTL (including system variables, system subroutines, and GPIB statements).

4.1 Elements of Statement Configuration

4.1.1 Line number

(1) Function

A line number is placed at the beginning of each statement and serves as an index during program editing or execution.

(2) Format

Numeric String
|
Integer constant from 1 to 65535

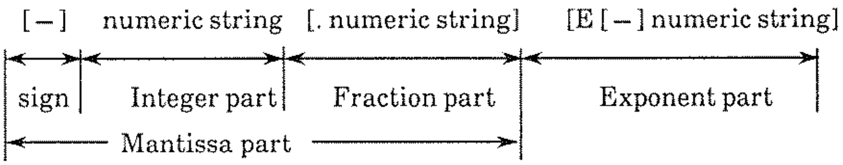
4.1.2 Constants

(1) Function

A constant represents a specific numeric value, character string or bit string.

(2) Format

(a) Numeric constants



The maximum number of mantissa digits is 15 (including a sign and a decimal point), and the range of exponent part is 10^{308} to 10^{-307}.
When a numeric constant is assigned to an integer type numeric variable, the range is −32768 to +32767.

(b) Character constants

"String"

|

1 to 255 characters enclosed with double quotation marks (" ")

Note: One line of program corresponds two lines on screen. Then, maximum number of characters on a program line is limited to the value.

(c) Bit constants

- Hexadecimal constant

  $ <u>Hexadecimal expression</u>

  |

  0 to FF

- Binary constant

  # <u>Binary expression</u>

  |

  0 to 11111111

---

(3) Examples

(a) Numeric constants

```
1
-12.3
12E3          ···Equal to 12000
-Ø.12E-3      ···Equal to −0.00012
```

(b) Character constant

```
"Who are you? "
```

(c) Bit constants

```
$F            ···Equal to #1111 (binary) or 15 (decimal).
#ØØØ11Ø1Ø     ···Equal to $1A (hexadecimal) or 26 (decimal)
```

4.1.3 Variables

There are simple, array, and system variables. For the system variable, refer to paragraph 5.1.

(1) Simple variable

There are numeric, character string, and bit string variables. The simple variable consists of eight or less characters, the first of which must be an upper-case alphanumeric character as shown below:

- Real number-type numeric-variable name: Upper-case alphabetic character [alphanumeric [alphanumeric]] ——— ABCD0123
- Integer-type numeric-variable name: Upper-case alphabetic character [alphanumeric [alphanumeric]] % ——— A%
- Character-string-variable name: Upper-case alphabetic character [alphanumeric [alphanumeric]] $ ——— ABC$
- Bit-string-variable name: Upper-case alphabetic character [alphanumeric [alphanumeric]] # ——— A#

Notes:

- Character-string simple variables can store up to 10 characters.
- Bit-string simple variables can store up to an 8-bit string.

(2) Array variable

The variable (declared as an array by the DIM statement) is called an array variable. Some system variables are also handled as array variables. The format of the array variables is shown below.

- Array variable : variable (numeric constant or numeric variable)

Subscript range
- For numeric-array variable 0 to 1023
- For character-string-array variable 0 to 254
- For bit-string-array variable 0 to 7

Notes:

- The subscript range for an array variable is from 0 to array size −1.
- When the subscript in the array variable is a real number, it is truncated after the decimal point.
- Up to 256 variables can be used (except for system variables).
- Pre-registered symbols (such as commands, statements, functions and system variables) cannot be used as user-defined variable names.
- For character-string-array variables, the maximum number of characters to be stored is declared as an array size. The range that can be obtained as an array is 1 to 255.
- For bit-string-array variables, the maximum number of bit strings is declared as an array size. The range that can be obtained as an arrange is 1 to 8.

4.1.4 Multi statement

By using '& ' as the delimiter in a statement, multiple statements can be entered on the same line.

This delimiter can also be used to enter a program of two lines. There are no restrictions on the number of statements within a program, provided that the length of the program does not exceed two lines.

Example :

```
1Ø  FOR I=Ø TO 1Ø & A=I*I & PRINT A & NEXT I
2Ø  STOP
```

4.1.5 Functions

There are basic functions (arithmetic, boolean, statistical and character-string functions) and dedicated functions in PTL. The system functions are used for measurement evaluation.

(1) Arithmetic function

Table 4-1 Basic Functions

Function name	Function	Parameter	
Sine	SIN(X)	The X unit is degrees.	A constant or a variable is used for X.
Cosine	COS(X)		
Tangent	TAN(X)	X≠ ±90(2n+1) , n:any integer	
Arcsine	ASN(X)	$\lvert X \rvert \leqq 1$	
Arccosine	ACS(X)		
Arctangent	ATN(X)		
Natural logarithm	LN(X)	$X > 0$	
Common logarithm	LOG(X)		
Exponent	EXP(X)		
Square root	SQR(X)	$X \geqq 0$	
Absolute value	ABS(X)		
Sign	SGN(X)	FOR X > 0 , SGN (X) =1 FOR X < 0 , SGN (X) = −1 FOR X = 0 , SGN (X) =0	
Integer value	INT(X)	X : Numeric type constant variable (An integer less than X is returned.)	
Rounding up	ROUND(X[,N])	X : Numeric type constant variable N : Numeric type constant variable (default value: N = 0) (X is rounded up to the N-th decimal place.)	
Function to calculate the quotient and remainder	Q=DIV(R,S,D)	Q : Numeric variable ---- Stores the quotient R : Numeric variable ---- Stores the remainder S : Numeric variable ---- Stores the dividend D : Numeric variable ---- Stores the divisor	
Function to isolate the integer and decimal parts of a real number	I=FIX(S,D)	I : Integer variable --Stores only the integer part S : Real-number variable -- Stores the real number of the original value D : Real-number variable -- Stores only the decimal part	

(2) Boolean functions

Table 4-2 Boolean functions

Function name	Function	Parameter
Negation	NOT(X)	X and Y are constants and variables of bit type or numeric type, and hexadecimal constants.
Logical product	AND(X,Y)	
Logical sum	OR(X,Y)	
Exclusive OR	EOR(X,Y)	

(3) Statistical functions

Table 4-3 Statistical functions

Function name	Function	Parameter
Function to find maximum value	MX=max(S)	S : Variable defined as one-dimensional array MX : Stores the maximum value MN : Stores the minimum value SM : Stores the sum total MS : Stores the mean value VR : Stores the variance $\text{Variance} = \frac{\Sigma (X-\overline{X})^2}{\text{No. of samples}}$
Function to find minimum value	MN=min(S)	
Function to find sum	SM=sum(S)	
Function to find mean value	MS=mean(S)	
Function to find variance value	VR=var(S)	
Function to find all above values	VR=sta (S,MX, MN,SM,MS)	

Note: The left side always consists of numeric variable in which found (calculated) value is stored. The one-dimensional S-parameter is valid even if there is only one element provided. When all the elements are to be processed statistically, no subscript is necessary at the entry. If a subscript is included, only the element specified by the subscript will be processed.

(4) Character-string functions

(a) Interchange between numerics and characters (strings)

1. **ASC(Alphabetic constant or variable)**
   ASC generates the character code for the first character of the string.

2. **CHR$(Constant or variable)**
   CHR$ generates the character with the character code corresponding to the parameter value. For a character type, the character remains unchanged. The parameter range is from 0 to 255.

3. **STRING$(Numeric constant or variable, constant or variable, character constant or variable)**
   STRING$ generates the characters (with the character code of the numeric value or the first character of string specified by the 2nd parameter) by the number of characters specified by the 1st parameters.Up to 255 repetitions may be specified.
   Refer to CHR$ ( )

4. **HEX$ (numeric-value-type constant or variable 1 [, numeric-value-type constant or variable 2])**
   A decimal value of the first parameter is given as a hexadecimal character string with number of digits specified by the 2nd parameter.
   An error will occur if the value of the first parameter does not fall in between -2^{31} and $2^{32}-1$. An error will occur if the second parameter goes beyond eight digits. When omitted, the return value will be of variable length.

5. **OCT$ (Constant or variable)**
   OCT$ generates the octal character string corresponding to the parameter value. An error is generated when the range -32768 to 32767 is exceeded.

6. **BIN$ (numeric-value-type constant or variable 1 [, numeric-value-type constant or variable 2])**
   A decimal value of the first parameter is given as a binary character string with number of digits specified by the 2nd parameter.
   An error will occur if the value of the first parameter does not fall in between -2^{31} and $2^{32}-1$. An error will occur if the second parameter goes beyond 32 digits. When omitted, the return value will be of variable length.

7. **CVI (Character constant or variable of 2 or more characters)**
   CVI generates the value converted from a character string to an integer numeric expression. If the character string exceeds two characters, the excess part is disregarded. Conversely, an error is generated when it is less than 2 characters..

8. **CVD (Character constant or variable of 8 or more characters)**
   CVD generates the value converted from a character string to a double-precision real-number numeric expression. When the character string exceeds 8 characters, the excess part is disregarded. Conversely, an error is generated when it is less than 8 characters.

9. **MKI$ (Integer constant or variable)**
   MKI$ generates the corresponding character code of the internal binary expression of the specified numeric value.
   This is the reverse process of the previously-mentioned CVI.

10. **MKD$ (Double-precision real-number constant or variable)**
MKD$ generates the corresponding character code of the internal binary expression of the specified numeric value.
This is the reverse process of the previously-mentioned CVD.

11. **VAL (Character variable, Number constant or variable 1, numeric constant or variable 2)**
VAL isolates the mth to nth numeric characters (including other than numeric code) of the specified data string and changes them to the double-precision real-number numeric expression, assuming that m and n are the specified values by variable 1 and variable 2, respectively.
Both m and n may be omitted. When m is omitted, the object runs from the head character of the data string: and when n is omitted, the object runs to the last character of the data string.
An error occurs when no numeric character is found.

12. **BVAL (character constant or variable)**
This function will convert the parameter string notated in binary into an unsigned decimal value.
An error will occur if the parameter exceeds 32 bits. All characters other than "0"or "1" will be ignored.

13. **HVAL (character constant or variable)**
This function will convert the parameter string notated in hexadecimal into an unsigned decimal value.
An error will occur if the parameter exceeds 32 bits (8 characters). Characters other than "0" to "9" and "A" to "F" are ignored.

14. **CHR (Numeric constant or variable)**
CHR generates the same character string as that to be displayed by the PRINT statement within the specified numeric value by parameter.

15. **STR$ (Numeric constant or variable)**
This performs exactly the same processing as described for the CHR function.

(b) Retrieving character strings

1. **INSTR ([Numeric constant or variable,] character constant or variable 1, character constant or variable 2)**
When character string 2 is found within character string 1, its position is returned; if it is not found, 0 is returned. When the numeric value is included in the 1st parameter, the search starts from the indicated position with the numeric value; when it is omitted, the search starts from the header. The range of the value is from 1 to 255.

2. **LEFT$ (Character constant or variable, numeric constant or variable)**
This gives the specified number of characters (counting from the left) as specified by the second-parameter. When the specified number exceeds the number of characters in the strings, whole the character string is given. The specifiable number is from 0 to 225. When the specified number is 0 , a null string is returned.

3. **MID$ (Character constant or variable, numeric constant or variable 1, numeric constant or variable 2)**
   This gives the n of character strings from the m-th character, assuming that the m and n are the specified values by the variable 1 and variable 2, respectively. The range of m/n is (1 to 256) / (1 to 255), respectively.
   When m exceeds the total number of characters, a null string is returned.

4. **RIGHT$ (Character constant or variable, numeric constant or variable)**
   This performs the same processing as the LEFT$ ( ) command but from the right side. The value range is also the same (0 to 255). Note that this command does not reverse the character string sequence.

5. **LEN (Character constant or variable)**
   LEN gives the number of characters in a character string including all character codes from 0 to $1F.

6. **SLEN (character type constant or variable)**
   This gives the number of characters composing a character string in the same manner as specifying a value in LEN ( ).
   However, this gives the length with the space at the end of the character string omitted in the variable.

7. **SGET$ (character type constant or variable)**
   This gives a valid character string with the space at the end omitted.

(5) Dedicated functions

Function description	Function	Parameter
Reads the error code and line number in which error occurred on	`V=ERRREAD(m)`	m 0 : Error code 1 : Line number in which error occurred
Reads the type of event	`A#=STATUS(m)`	m 0 : Event 0 1 : Event 1 2 : Event 2 3 : Event 3
Reads the date and o'clock, minute, second	`A$=DTREAD$(m)`	m 0 : Date (YY-MM-DD) 1 : o'clock, minute, second (HH:MM:SS)
Random number generation (more than 0, less than 1)	`RND(m)`	m : Specify an arbitrary value.

Notes:

- ERRREAD (m) can only be used during at error interrupt. For details on error interrupts, see Paragraph 4.2.36.
- STATUS (m) can only be used during an event interrupt. For details on event interrupts, see Paragraph 4.2.23.
- m is a numeric constant or numeric variable.
- The sequence of pseudo-random numbers generated by RND (m) becomes the same each time RUN is executed.
  See Paragraph 4.2.54, " RNDMIZE statement " for how to change the sequence.

4.1.6 Arithmetic operators

(1) Function

These operators perform addition, subtraction, multiplication, division, and exponential operations.

(2) Format

=	···	Substitution
+	···	Addition
−	···	Subtraction
*	···	Multiplication
/	···	Division
!	···	Exponentiation
( )	···	Represents operation priority (Operations in parentheses are performed first.)

(3) Operation Priority

The operation priority is shown below.

Operation priority	Arithmetic operators
High	!
↕	* /
	+ −
Low	=

Notes:

- Bits and characters cannot be used in operations.
- If X of X ! Y is a minus number, but Y is a plus number, X ! Y can be operated.
- If there is a different type variable on the right side of an equals sign (=), an overflow or underflow error may occur.
- Number of digits of divided becomes number of digits of the solution on division with numerals or variables.

(4) Example

```
A$="abc"
C=(D+100)/E
J=((K+1)*10-M)*10
```

4.1.7 Relational operators

(1) Function

These operators perform relational operations.

(2) Format

=	···	Equal (=)
>< or <>	···	Not equal (≠)
>	···	Greater than (>)
<= or =<	···	Equal to or less than (≦)
<	···	Less than (<)
>= or =>	···	Equal to or greater than (≧)

(3) Comparing character strings

When comparing the sizes of character strings, count only significant characters.
(Ignore any spaces at the ends of the character strings to the left and right of an operator)

- If two character strings are the same length, their characters are compared sequentially from the beginning. The first character which is different is found. The character which has the lower code value will determine the smaller character string.

  Example : ABC is smaller than ABX.

- If two character strings are different lengths, the character strings over their common length are compared. If the two strings are equal over this length, the shorter character string will be the smaller character string.

  Examples : ABX is larger than ABCD.
  ABC is smaller than ABCD.

- The smallest character string is one with 0 length.

  Example : The length of A$ is 0 when DIM A$ (10) is declared.

(4) Examples

```
IF C=Ø GOTO 1ØØ
IF JKL>=168 STOP
```

4.1.8 String concatenation (the " + " operator)

(1) Function

String concatenation is possible with the "+" operator.

(2) Format

$$\left\{\begin{array}{l}\text{character string constant}\\ \text{character string variable}\\ \text{character string function}\end{array}\right\} + \left\{\begin{array}{l}\text{character string constant}\\ \text{character string variable}\\ \text{character string function}\end{array}\right\}$$

Notes:

- Only be used with the right hand parameter of the LET statement.
- You cannot concatenate character string and numeric values, character string and bit, or bit and bit.

(3) Examples

```
100 A$="ABC"
110 B$="DEF
120 A=INSTR(A$,"␣")-1
130 B=INSTR(B$,"␣")-1
140 C$=LEFT$(A$,A)+LEFT$(B$,B)
150 PRINT "A$=",A$
160 PRINT "B$=",B$
170 PRINT "C$=",C$
```

```
AS=ABC␣␣␣␣␣␣␣
B$=DEF␣␣␣␣␣␣␣
C$=ABCDEF␣␣␣␣
          └──── Space
```

Notes:

- Simple character-string variables are assumed to be a ten-character array-declared variables, implicitly. Therefore, characters not assigned will be filled with spaces. For details, see Paragraphs 4.2.14 and 4.2.15.
- By using the above method, you can concatenate actual stored character only.

4.1.9 Formats

(1) Function

These formats specify the format of strings in output operations. Integers, real numbers without exponents, real number with exponents, strings, binary numbers, and hexadecimal numbers can be specified.

(2) Formats

- Integer

  : I number of digits

  (1 to 18)

- Real number without exponent

  : F number of all digits. number of fractional digits

  (4 to 20)

  (Number of all digits ≧ number of fractional digits + 3)

- Real number with exponents

  : E number of all digits. number of fractional digits

  (9 to 24)

  (Number of all digits ≧ number of fractional digits + 8)

- String

  : C number of digits

  (0 to 255)

- Binary number

  : B number of digits

  (1 to 8)

- Hexadecimal number

  : H number of digits

  (1 or 2)

(3) Examples

```
PRINT A$:C3,J:F1Ø.4
```

Notes:

- When number of digits is 0 for string, the character length becomes variable to output all actual length of the character string variable.
- A single space is included at the end of each PRINT statement provided that the FORMAT specifiers are capitalized. These spaces can be omitted by using a small-case FORMAT specifier instead of a capitalized FORMAT specifier (See paragraphs 4.2.14 and 4.2.15.)

4.1.10 Label

(1) Function

A jump address can be assigned indirectly by using a label with a line number in a statement such as GOTO or GOSUB.

(2) Format

```
Line number△*label
Line number△*label△statement
```

- A label consists of up to eight alphanumeric characters starting with an upper-case alphabetic character. The label is prefixed with *.
- When multiple line numbers are defined with the same label, an error occurs during program execution.

(3) Examples

```
 10 INPUT A
 20 IF A=0 GOSUB *ABC1
 30 IF A<>0 GOSUB *ABC2
 40 GOTO 10
100 *ABC1
110 PRINT "OK!"
120 RETURN
200 *ABC2
210 PRINT "NG!"
220 RETURN
```

4.2 Basic Statements

4.2.1 Comment (REM statement)

(1) Function

This statement gives comments to program. These comments are not executed by the system and they have no effect on program execution.

Note: When a specific statement is described as a comment statement, it must be enclosed by a pair of double quotation marks(" ") as a character constant.

(2) Format

```
REM ["comment"] or
' [comment]
```

(3) Examples

```
1Ø  REM
2Ø  REM "Compute average"
3Ø  'Compute average
4Ø  A=1ØØ 'Initial set
```

4.2.2 Array declaration (DIM statement)

(1) Function

This statement declares arrays. Arrays must be one-dimensional or two-dimensional, and are restricted at a size as shown in paragraph (2) below according to the type of variable name.

(2) Format

```
DIM variable-name (array-size[, array-size])
    [,variable-name (array-size[,array-size])....]
```

Notes:

- The same variable name cannot be redefined as an array. A variable (that has been used as an independent variable) cannot be declared as an array.
- Error W225 will be generated when a two-dimensional array is referred to without the specification of two dimensions.
- Error W224 will be generated when a one-dimensional array is referred to as a two-dimensional array.

- The size limit of the declarable array is as follows. If the declared size exceeds these limits, ERROR 203 will be generated.

Character type ..	1 to 255	Two dimensional array:		
Bit type	1 to 8	One dimensional side	Two dimensional side	
Numeric type ...	1 to 1024	1 to 1024	Character type ...	1 to 255
			Bit type	1 to 8
			Numeric type	1 to 1024

- For the numeric type, the program area will become insufficient; thus, it is impossible to define 1024 on both the one- and two-dimensional sides. In this case, ERROR 206 will be generated.
  The total number of array elements that can be declared (product of the number of one-dimensional array elements by the number of two-dimensional array elements) is not restricted because it depends on the capacity of empty memory.
- For the character array, ten characters long are automatically declared when no array is declared.
- For the bit type, array eight bits long are automatically declared when no array is declared.
- Error W224 occurs when individual elements are referred to (read or written) without the appropriate array declaration.

(3) Examples

```
DIM CARR(1ØØ) , A$(5,12)
DIM I#(Ø),  ALP$(4Ø)
```

(4) System variables which have been unconditionally declared as arrays.

```
XMA(*), XMB(*), XMC(0 to 624, 0 to 1), XMD(0 to 4319), XME(0 to 250),
XMT(*), SMA(*), SMB(*), IMA(*), IMB(*), RMA(*), RMB(*), IOA(*1),
IOB(*1), IOC(*2), IOD(*2)
```

Notes:

* is an array element of 0 to 1001.

*1 is an array element of 0 to 7. (It is valid when the I/O port interface is used.)

*2 is an array element of 0 to 3. (It is valid when the I/O port interface is used.)

4.2.3 Initialization (CLEAR statement)

(1) Function

Initializes user-defined variables.

(2) Format

```
CLEAR
```

Note: When the CLEAR statement is executed, the array can be redefined since variables are re-initialized in a manner similar to that in which executing RESET is executed.

4.2.4 Substitution (LET statement)

(1) Function

This statement substitutes variables for constants, variables, and results of operations. See paragraph 4.1.6 for the arithmetic operators.

(2) Format

$$[\text{LET}]\ \text{variable}=[\ (\ ]\left\{\begin{array}{l}\text{constant}\\ \text{variable}\\ \text{function}\end{array}\right\}[\ )\ ]$$

$$[\text{arithmetic } \underset{\begin{array}{c}|\\ +\ \ -\\ *\ \ /\\ !\end{array}}{\text{operator}}\ [\ (\ ]\left\{\begin{array}{l}\text{constant}\\ \text{variable}\\ \text{function}\end{array}\right\}[\ )\ ]\ldots]$$

$$[\ \text{LET}\ ]\ \text{character type variable} = \left\{\begin{array}{l}\text{character string constant}\\ \text{character type variable}\\ \text{character string function}\end{array}\right\}+$$

$$\left\{\begin{array}{l}\text{character string constant}\\ \text{character type variable}\\ \text{character string function}\end{array}\right\}+\ \ldots$$

Notes:

- Bits and characters cannot be used in operations.
- If a substitution statement is placed after an IF statement, LET cannot be omitted.

(3) Examples

```
LET A=B+C or A=B+C
IF X=Ø LET Y=1Ø
```

4.2.5 Branch (GOTO statement)

(1) Function

This statement changes the sequence of program execution to the statement of the specified line number.

(2) Format

```
GOTO line number or GOTO *label
```

4.2.6 Termination of execution (STOP statement)

(1) Function

This statement terminates program execution after displaying an execution termination message on the CRT screen as follows.

STOP IN line number

(2) Format

```
STOP
```

Note: Suspension specifications are ignored in STOP statements, since program execution is terminated.

4.2.7 Branch to subroutines (GOSUB statement)

(1) Function

This statement changes the program execution to the subroutine with the specified line number. When the RETURN statement is executed at the end of the subroutine, the program execution is returned to the statement following the GOSUB statement.

(2) Format

```
GOSUB line number or GOSUB *label
```

Note: Calling another subroutine during execution of a subroutine is referred to as "nesting". Up to 10 nesting levels are permitted.

4.2.8 Return from subroutines to main routine (RETMAIN statement)

(1) Function

When the RETMAIN command is used during program execution, control is returned to the highest level of the routine regardless of the nesting level.

(2) Format

```
RETMAIN
```

Note: If the RETMAIN command has been executed in the highest level of the routine, ERROR F213 occurs.

4.2.9 Return from subroutines (RETURN statement)

(1) Function

This statement returns program execution from the subroutine to the statement following the corresponding GOSUB statement.

(2) Format

```
RETURN
```

4.2.10 Decision (IF statement)

(1) Function

If the result of the relational operation is true, this statement executes the subordinate statement. For relational operators, see paragraph 4.1.7.

(2) Format

```
       ┌constant ┐                      ┌constant ┐
IF     {         }  relational operator {         }  statement
       └variable ┘                      └variable ┘
                            ↓
                          =
                          >< or <>
                          >
                          <= or =<
                          <
                          >= or =>
```

Notes:

- All statements including IF statements can be placed as subordinate statements.
- Relational operations can not be performed among numerical values, characters, and bits.
- If a substitution statement is placed after an IF statement, LET cannot be omitted.

(3) Examples

```
IF C=1 GOTO 1ØØ
IF ACH$=BCH$ PRINT ACH
IF C<1Ø IF C>=2Ø PRINT "ERROR"
IF C<1Ø LET C=1Ø
```

4.2.11 Repetitions start (FOR statement)

(1) Function

This program loop causes the program code (located between the FOR and NEXT) to be repeatedly executed, until the specified variable is equal to or greater than the specified end value.

Up to 10 nesting levels may occur within a FOR statement.

(2) Format

```
                              ┌numeric constant ┐      ┌numeric constant ┐
FOR numeric variable = {                  } TO {                  }
                              └numeric variable ┘      └numeric variable ┘
                                      |                         |
                                Initial value             Ending value

   ┌numeric constant┐
[STEP {                } ]
   └numeric variable┘
           |
  Increment (default value is 1)
```

Notes:

- Even if the initial value exceeds the end value, one operation cycle will be performed.
- NEXT statements may be used anywhere; however, for proper execution they must be properly positioned.

(3) Example

```
FOR C=1 TO 100                    ┐
FOR T=TB TO TE STEP 0.1         ┐ │
FOR D=-1 TO -10 STEP -1       ┐ │ │
NEXT D                        ┘ │ │  Repeats
NEXT T                          ┘ │
NEXT C                            ┘
```

4.2.12 Repetition termination (NEXT statement)

(1) Function

This statement is used with its corresponding FOR statement to terminate the repeated operation.

(2) Format

```
NEXT numeric variable
           |
Same variable as that specified in FOR statement
```

4.2.13 Key-input (INPUT statement)

(1) Function

This statement is used to assign data input from the external keyboard or the front panel key to variables.
When the statement is executed, the following message is displayed on the CRT.

? ▨

Input data after the displayed question mark ? via the numeric key of the keyboard or the front panel, then press the ⏎ key or [ENTR] key of the instrument.

Use commas (,) as delimiters of data if required.

(2) Format

```
INPUT ["displayed character string",] variable[,variable....]
```

Notes:

- If a real number is input for an integer variable, it is truncated under decimal point.
- If the input data length is smaller than that which has been declared, spaces are appended to the entry. If it is greater, the excess digits will be truncated.
- For numeric and bit type variables, spaces before and after the input value are ignored.
- Hexadecimal data cannot be input.
- Five variables can be specified .
- The ,(comma) and – (minus) are input by pressing the [kHz] key and the [MHz] key of the front panel, respectively.

(3) Examples

```
INPUT "COUNT=",C   → COUNT=? 123
INPUT C,A$,I#      → ? 123,Q,101101
```

4.2.14 Display (PRINT statement)

(1) Function

This statement edits and displays data on the CRT screen.
Unformatted data is displayed with spaces added after its effective digits. The format name and output formats are shown in Table 4-4.
For the format, see paragraph 4.1.9.
Line feed is disabled by adding " ; " at the end.

Table 4-4 Format Name and Output Format

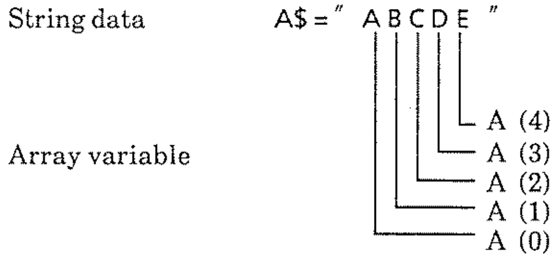
Format name	Output format
I	Zero-suppressed integer (Ex. ␣␣ 123)
F	Zero-suppressed integer and zero-suppressed fraction (Code digit exists.) (Ex. ␣␣123.45␣)
FP	Zero-suppressed integer and zero-suppressed decimal number (unsigned) (Ex. ␣123.45␣)
E	$\begin{Bmatrix} \sqcup \\ - \end{Bmatrix}$ Zero-suppressed fraction E [−] exponent (Ex. ␣1.23E − 2␣)
C	String ··· If the size of data is smaller than the specified format size, spaces are added; and if it is greater, the excess lower digits are truncated.
B/H	Zero-suppressed binary-number / hexadecimal-number string (Ex.␣␣ 1011)

(2) Format

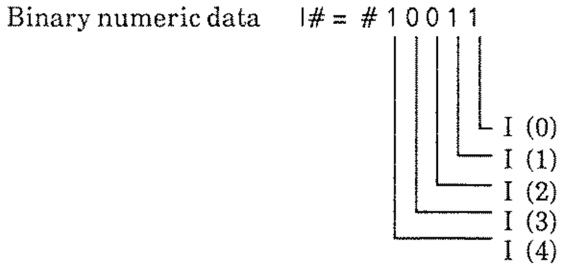
```
         ┌ variable [:format] ┐    ┌ variable [:format] ┐
PRINT {                       } [,{                      } ···] [ ; ]
         └ string constant    ┘    └ string constant    ┘         |
                  |                                               |
          Constant displayed as is                         No line feed
```

Notes:

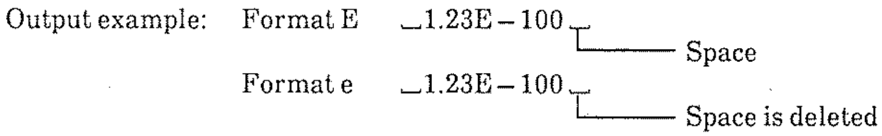
- Up to five variables or constants can be specified.
- Values which cannot be expressed are displayed as ***...*.
- A string-which is an array of character variables- is comprised as follows:



- A binary numeric variable - which is an array of binary digits - is comprised as follows:



- The last space can be deleted by using a lower-case format i, f, fp, e, c, b, or h instead of an upper-case format I, F, FP, E, C, B, or H.



- Only plus values are significant in format FP.

(3) Data and print output examples

Table 4-5 shows data and output examples.

Table 4-5 PRINT-Statement Output Example

Format	Data	Statement	Output
(None)	T=1234.45	PRINT␣T	123.45␣
	A$="ABCD"	DIM␣A$(5) PRINT␣A$ PRINT␣A$(2)	 ABCD␣␣ C␣
	A$(Ø,)="AB" A$(1,)="CD" A$(2,)="EF"	DIM␣A$(3,2) PRINT␣A$(1,Ø) PRINT␣A$(2,)	 C␣ EF␣
I	T=1234.56	PRINT␣T:I6 PRINT␣T:I4 PRINT␣T:I3	␣␣1234␣ 1234␣ ***␣
F	T=-123.45	PRINT␣T:F6.1 PRINT␣T:F9.2 PRINT␣T:F9.3	-123.4␣ ␣␣-123.45␣ ␣-123.45Ø␣
	T=123456	PRINT␣T:F9.1 PRINT␣T:F5.1	␣123456.Ø␣ *****␣
FP	T=123.45	PRINT␣T:FP6.1 PRINT␣T:FP9.2 PRINT␣T:FP9.3	␣123.4␣ ␣␣␣123.45␣ ␣␣123.45Ø␣
	T=123456	PRINT␣T:FP9.1 PRINT␣T:FP5.1	␣123456.Ø␣ *****␣

E	T=-123.45
	
	T=-Ø.12E1
		PRINT␣T:E1Ø.2
		PRINT␣T:E13.5
		PRINT␣T:E15.7
		PRINT␣T:E9.2
			-1.23E2␣␣␣␣
			-1.2345␣E2␣␣␣␣
			-1.2345␣␣␣E2␣␣␣␣
			-1.2␣EØ␣␣␣␣

C	A$="F"	PRINT␣A$:C3	F␣␣␣
	A$="ABCDE"	DIM␣A$(5) PRINT␣A$:C7 PRINT␣A$:C3 PRINT␣A$:C5 PRINT␣A$(3):C3	 ABCDE␣␣␣ ABC␣ ABCDE␣ D␣␣␣
	A$="ABCDEF"	DIM␣A$(6) PRINT␣A$ PRINT␣A$(3)	 ABCDEF␣ D␣

Table 4-5 PRINT-Statement Output Example (Continued)

Format	Data	Statement	Output
B	I#=#1	PRINT␣I#:B1 PRINT␣I#:B3	1␣ ØØ1␣
	I#=#1Ø11	DIM␣I#(4) PRINT␣I#:B5 PRINT␣I#:B3 PRINT␣I#(3):B3 PRINT␣I#(Ø):B1	 1Ø11␣␣ Ø11␣ 1␣␣␣ 1␣
	I#=#1 I#=#1Ø11	PRINT␣I# DIM␣I#(4) PRINT␣I#	␣␣␣␣␣␣␣1␣ 1Ø11␣
	I#=#1ØØ11Ø1Ø	DIM␣I#(8) PRINT␣I# PRINT␣I#(3)	 1ØØ11Ø1Ø␣ 1␣
	I#=#ØØØ1ØØ11	PRINT␣I#	␣␣␣1ØØ11␣
H	I#=#1	PRINT␣I#:H1 PRINT␣I#:H2	1␣ ␣1␣
	I#=#1Ø1Ø	DIM␣I#(4) PRINT␣I#:H1 PRINT␣I#:H2	 A␣ A␣␣
	I#=#ØØØØ1Ø1Ø	DIM␣I#(8) PRINT␣I#:H1 PRINT␣I#:H2	 A␣ ␣A␣
	I#=#111Ø1Ø1Ø	DIM␣I#(8) PRINT␣I#:H1 PRINT␣I#:H2 PRINT␣I#(3):H1 PRINT␣I#(3):H2 PRINT␣I#(4):H1 PRINT␣I#(4):H2	 A␣ EA␣ 1␣ 1␣␣ Ø␣ Ø␣␣
	I#=#ØØ11ØØ	DIM␣I#(6) PRINT␣I#:H2	 ␣C␣
	I#=#11ØØ1Ø	PRINT␣I#:H2	32␣

Note:

Example with the DIM statement means the array declaration is performed for the variable. If no DIM statement is marked, it means there is no array declaration for the variable.

4.2.15 Reverse display (PRINTR statement)

(1) Function

Edits data and displays the data on the screen in reverse mode.
See Paragraph 4.2.14, " PRINT statement " for details.

(2) Format

```
         ┌variable [ : format]     ┐      ┌variable [ : format]     ┐
PRINTR  {                           } [, {                           } ···][;]
         └character-string-constant┘      └character-string-constant┘      |
                  |                                                         |
          The constant is displayed as is.                           No line feed
```

Notes:

- Only characters of character codes 0 to 127 can be displayed in reverse mode. PRINTR containing other character displays has the same function as that of PRINT. In this case, PRINTR displays characters in normal mode.
- A line in which characters of character codes 128 to 255 are displayed cannot be displayed in reverse mode. In this case, PRINTR has the same function as that of PRINT, and it displays characters in normal mode.
- If characters are displayed in reverse mode on the line reversed by PRINTR, reverse is canceled.

4.2.16 Positioning the cursor (LOCATE statement)

(1) Function

This statements specifies the cursor position on the screen. (Referred to at the upper left on the screen)

(2) Format

```
LOCATE (m, n)
  m → column position(1 to 68)
  n → line position(1 to 25)
```

Note: Both m and n are numeric constants or variables.

4.2.17 Data statement (DATA statement)

(1) Function

This statement defines numeric, bit and character constant to be read with the RDATA statement.

(2) Format

```
DATA, constant, constant,········
```

Note: Any number of parameters maybe input in a DATA statement provided that it does not exceed two lines. Further, different types of constants may be input in a single DATA statement.

4.2.18 Reading data (RDATA statement)

(1) Function

This statement reads values from the DATA statement and assigns them to variables.

(2) Format

```
RDATA variable, variable,········
```

Notes:

- Any number of parameters maybe assigned in an RDATA statement provided that it does not exceed 2 lines. Further, different types of constants may be input in a single RDATA statement.
- If the definition type in the DATA statement and the type of the substituted variable are incompatible at data reading with the RDATA statement, ERROR W208 will be generated.

4.2.19 Read specification of data statement (RESTORE statement)

(1) Function

This statement specifies the data statement to be read with the RDATA statement.

(2) Format

```
RESTORE [line number or *label]
```

Example :

```
  100  RESTORE 1ØØØ
  11Ø  FOR I=Ø TO 1Ø
  12Ø  RDATA A (I)
  13Ø  NEXT I
        ⋮
 1ØØØ  DATA Ø, 1, 3, 7, 9, 11, 13, 17, 19, 23, 29
```

Note: When the RESTORE-statement parameter is omitted, the first data statement is used.

4.2.20 Setting measurement parameters (PUT and WRITE 1000 statements)

(1) Function

Sets the MS8604A measurement parameters from the PTA.
The same messages as setting from GPIB are used.
This command is also used when sending inquiry messages to the MS8604A.

(2) Format

```
PUT character constant or character variable
WRITE 1000, variable or character constant [,variable or character
constant]
```

① PUT statement

- A message of the same format as when the MS8604A is controlled from GPIB is described in the operands.
- Only a character constant or character variable can be described in the operands.
- Only one constant or variable can be described.
- The format cannot be specified.
- When a fixed value is set at all times, the program can be simplified using this statement.

Examples :

```
PUT " FREQ 500MHZ"
```

→ Set measurement parameter center frequency to 500 MHz.

```
PUT " FREQ?"
```

→ Send measurement parameter center frequency inquiry message.

② WRITE 1000 statement

- A message of the same format as when the MS8604A is controlled from GPIB is described in the operands.
- Variables or character constants can be described in the operands.
- Up to five constants or variables can be described.
- When variables are used, the format can be specified.
- This statement is effective when setting is performed several times with only part of the control message being changed and when values treated as variables are set values in the program.

Examples :

```
F=500
WRITE 1000,"FREQ ",F,"MHZ"
```

→ Set measurement parameter center frequency to 500 MHz.

```
WRITE 1000,"FREQ?"
```

→ Send measurement parameter center frequency inquiry message.

4.2.21 Measurement parameter/data read (GET, COM and READ 1000 statements)

(1) Function

Reads the MS8604A measurement parameters and the measured result from the PTA.
The same messages as when the MS8604A is set from GPIB are used.

(2) Format

```
GET "inquiry command?",input variable
COM "inquiry command?">input variable[, input variable]
READ 1000, input variable[, input variable] or
READ 1000, input variable[;]
```

① GET statement

- An inquiry command can be sent and the response data can be read with one statement. Only one inquiry command can be described in one statement.
- Only a character constants or character variables can be described in the "inquiry command" parameters. Only one constant or variable can be specified. The format cannot be specified.
- The response data is stored in the input variable. When the response data contains a character, a character variable is specified. When the response data is numeric (numeric character) only, it may be a numeric variable or a character variable.
- When the response data consists of multiple data separated by a ",", everything up to the last data is stored in one variable as one data. Therefore, when a character variable is specified, if the array size is too small, all the response data may not be stored.
- Only one input variable can be specified. A ";" cannot be specified at the end of the statement.
- When the same inquiry command is always sent, the program can be simplified using this statement.

Example :

```
GET "FREQ?", A$
```

→ Send the center frequency inquiry message and store the response data in input variable A$.

② COM statement

- An inquiry command can be sent and the response data can be read with one statement. However, only one inquiry command can be described in one statement.
- Character constant or character variable or character constant and character variable can be specified in the "inquiry command" parameter.
  The format can also be specified for variables.
- The response data is stored in the input variable. When the response data contains a character, a character variable is specified. When the response data is numeric (numeric character) only, it can be a numeric variable or character variable.
- Multiple variables can be described. When the response data consists of multiple data delimited by a ",", the delimited data are stored sequentially in the specified variables.
  However, array variables cannot be used as input variables.
- A ";" cannot be specified at the end of the statement.
- This statement is effective when reading is performed several times with only part of the inquiry message changed and when sending an inquiry message for a value treated as a variable in the program.

Example :

```
I=1
COM "TEMPLVL? ", I>ML
```

→ Send the 1st marker of the template level inquiry message and store the response data in input variable ML.

Note: The inquiry message for each level of the template is specified by " TEMPLVL? n ". (n: template limit line No.) This statement is effective when the level of each template point is read by changing only the value of n.

③ READ 1000 statement

- This statement reads the response data only. Therefore, it is effective only when a PUT or WRITE 1000 statement is used to send an inquiry message.
- The response data is stored in the input variable. When the response data contains a character, a character variable is specified. When the response data is numeric (numeric character) only, it can be a numeric variable or character variable.
- Multiple input variables can be described. When the response data consists of multiple data delimited by a ",", the delimited data is stored sequentially in the specified variables.
- When the response data is treated as one data, even when it consists of multiple data delimited by a ",", the entire response, including the ",", can be stored in one variable by specifying ";" at the end of the statement. In this case, only one input variable can be specified. Data delimited by a "," can also be read by specifying only one variable without a ";" at the end and executing this statement repeatedly.
- When there is no response data, "***" is output.

Example :

```
WRITE 1ØØØ,"FREQ? "
READ 1ØØØ, A$
```

→ Store the response data to the center frequency inquiry command in A$.

4.2.22 Program loading and execution (CHAIN statement)

(1) Function

This statement loads and executes a file in PMC.

(2) Format

```
CHAIN "file name"
```

Note: The RUN, CONT or STEP commands (set in the execution state) remain valid even after the CHAIN command is executed. Consequently, the lines at which execution is suspended also remain effective.

4.2.23 ENABLE EVENT statement

(1) Function

Enables the specified I/O interrupt.

When the specified I/O interrupt occurs, the program will branch to the event interrupt subroutine defined by the ON EVENT statement.

(2) Format

```
ENABLE EVENT I/O number, event 3, event 2, event 1, event Ø
```

Notes:

- There are 2 types of I/O numbers: numeric variables and numeric constants.
- Events 0 to 3 can be numeric variables and constants, bit variables and constants, or hexadecimal constants.
- This statement can be executed directly.
- Events 0 to 3 indicate 32 bits of I/O interrupt events as shown below.
- The defined bits (b0 to b31) are enabled when "1" and disabled when "0".
- When the master bit (b31) was set to "1", all the defined conditions are valid regardless of the value of bits b0 to b30.

b31 b24	b23 b16	b15 b8	b7 bØ
Event 3	Event 2	Event 1	Event 0

(3) Types of I/O interrupts

(a) Time-specification interrupts

Three kinds of time-specification interrupts are available.

① DELAY

Generates an event interrupt after the specified time has elapsed.
The time can be specified as a GPIB command or by a PUT or WRITE statement.

DELAY setting

"EDLY t" t: 0 to 3600 (s) 1 sec resolution

- Time counting starts from the time set by this command.
- When the time is reset during counting, counting restarts.
- If t = 0 was set, counting is interrupted.
- There is no set value t inquiry command.

② Time

Generates an event interrupt at the specified time.
The time can be specified as a GPIB command or by a PUT or WRITE statement.

Time setting
"ETIM t1, t2, t3"

t1: Specifies the hour. (0 to 23)
t2: Specifies the minute. (0 to 59)
t3: Specifies the second. (0 to 59)

- When the time is reset during counting, counting restarts.
- There are no set value t1, t2, and t3 inquiry commands.

③ Cycle

Generates an event interrupt at the specified cycle (time).
The cycle can be specified as a GPIB command or by a PUT or WRITE statement.

Cycle setting

"ECYC t" t: 0 to 3600 (s) 0.1 sec resolution

- If t = 0 was set, time counting is interrupted.
- There is no set value t inquiry command.

(b) Soft keys and data knob interrupt

① Soft keys ([F1] to [F5])

When a PTA menu (3/4) [F1] to [F5] key (corresponding to system variables EX1 to EX5) is pressed, an event interrupt is generated. This also applies to the PTA keyboard [F1] to [F5] keys.

② Cursor control keys

When the PTA menu (2/4) [CURSOR UP : F2] key or [CURSOR DOWN : F3] key is pressed, an event interrupt is generated. This also applies to the PTA keyboard [↓] and [↑] keys.

③ Data knob

When the data knob is turned, an event interrupt is generated.
However, when MS8604A measurement parameter setting is effective, an event interrupt is not generated.

Clockwise and counterclockwise revolution can be detected.

(c) GPIB interrupt

When serial polling is received in device mode and SRQ (Service Request) is received in system controller mode, an event interrupt is generated.

① Serial polling received

This can be used only at the GPIB1 port. When the GPIB1 port is in device mode and SRQ (Service Request) is sent to the host computer and serial polling is then performed, an event interrupt is generated as soon as SRQ is turned off.

② SRQ received

This can be used with the GPIB1 port and GPIB2 port. However, when it is executed at the GPIB1 port, the port must be in system controller mode.
When a peripheral device issues a service request to the system controller, an event interrupt is generated. In the service request ON state, an event interrupt is not generated even if this event is enabled.

(d) RS-232C interrupt

This function can be used only when the RS-232C (option 02) interface is connected.
If an interrupt is received from the RS-232C port, an event interrupt occurs.

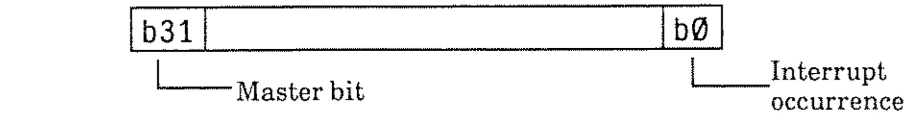
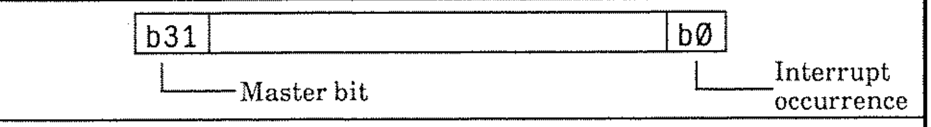
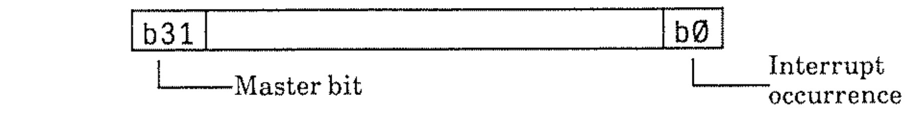
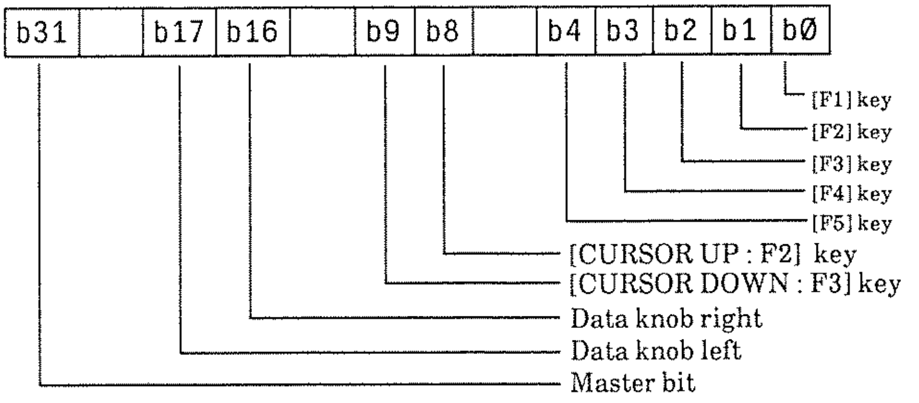
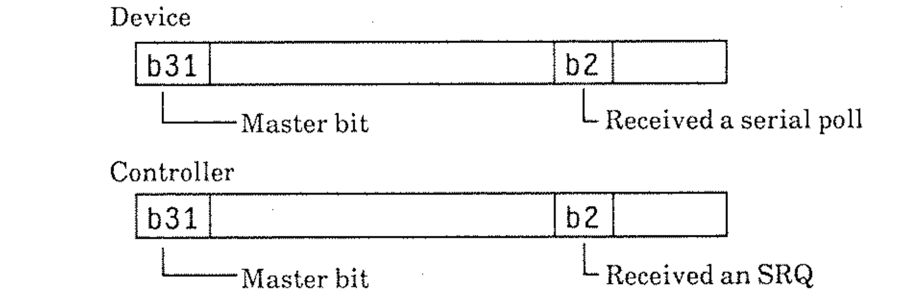
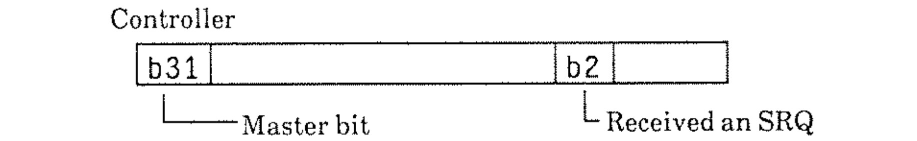
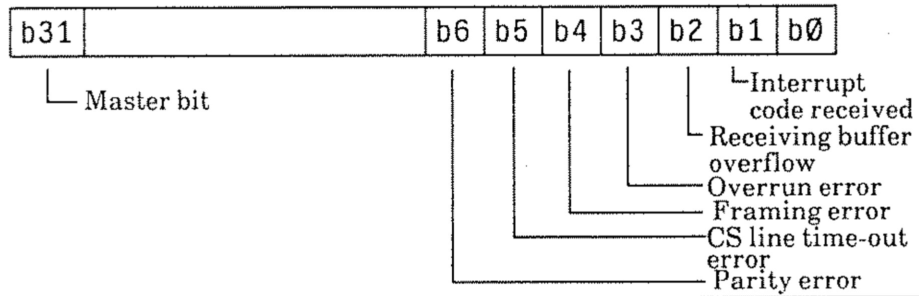
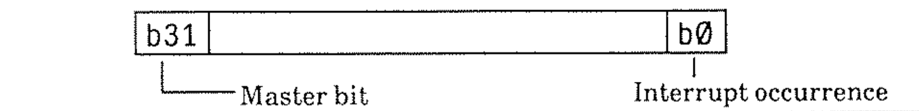
(e) I/O port interrupt

When a hardware interrupt is received from an I/O port, an event interrupt is generated. This event can be enabled and disabled independently from the IOEN, IOMA, and IODI statements.

The types of I/O interrupts, the I/O numbers, and the bits corresponding to each event are shown below.

Notes:

- There are 0 to 99 types of I/O.

I/O type	I/O number	Contents
Clock (DELAY)	1	
Clock (TIME)	2	
Clock (CYCLE)	3	
SOFT KEY, data knob	11	 b31 b17 b16 b9 b8 b4 b3 b2 b1 bØ [F1] key [F2] key [F3] key [F4] key [F5] key [CURSOR UP : F2] key [CURSOR DOWN : F3] key Data knob right Data knob left Master bit
GPIB (1) port	21	 Device b31 b2 Master bit Received a serial poll Controller b31 b2 Master bit Received an SRQ
GPIB (2) port	22	
RS-232C port	33	 b31 b6 b5 b4 b3 b2 b1 bØ Master bit Interrupt code received Receiving buffer overflow Overrun error Framing error CS line time-out error Parity error
I/O port	41	

4.2.24 DISABLE EVENT statement

(1) Function

Disables the specified I/O interrupt.

(2) Format

```
ENABLE EVENT I/O number[,event 3,event 2,event 1,event Ø]
```

Notes:

- There are 2 types of I/O number: numeric variables and numeric constants.
- Events 0 to 3 can be numeric variables and constants, bit variables and constants, or hexadecimal constants.
- Events 0 to 3 may be omitted. When omitted, all interrupt events will be disabled.
- This statement can be directly executed.
- The defined bits are disabled when "1" and retain their previous enable/disable state when "0". However, master bit (b31) setting is meaningless. (Don't care)

4.2.25 ON EVENT statement

(1) Function

Registers the subroutine to branch to when the specified I/O interrupt event occurs.

(2) Format

```
ON EVENT I/O number, line number (or *label)
```

Notes:

- There are 2 types of I/O number: numeric variables and numeric constants.
- This statement can be executed directly.
- The function STATUS (M) is used as the interrupt event identifier. For more details, see Paragraph 4.1.5 (5).
- When an I/O port generates an interrupt when there is an ON IO GOTO (GOSUB) statement, both this statement and the previous I/O port statement are executed. However, this statement is executed first.

4.2.26 RETINT statement

(1) Function

Returns from the event interrupt subroutine.

(2) Format

```
RETINT
```

Notes:

- If any other return command is executed to return from an event interrupt subroutine, an execution termination error (F243) will be generated.
- If the RETINT command is executed for other than event interrupt, an execution termination error (F251) will be generated.
- It is possible to branch to a normal subroutine (GOSUB ··· RETURN) from the event interrupt subroutine.

4.2.27 IOEN statement

(1) Function

This function enables an interrupt to be issued via the I /O ports. When the IOEN statement is executed, the program branches to the line number defined in the ON IO GOTO statement or the ON IO GOSUB statement.

(2) Format

```
IOEN
```

4.2.28 IODI statement

(1) Function

This statement disables all interrupts from the I /O ports.
When the IODI statement is executed, all definitions specified by the ON IO GOTO statement or ON IO GOSUB statement are ignored.

(2) Format

```
IODI
```

4.2.29 IOMA statement

(1) Function

This statement masks interrupts from the I/O ports. When the IOMA statement is executed, the definitions specified by the ON IO GOTO statements or ON IO GOSUB statements are ignored. When the IOEN statement is executed after an interrupt has been issued from an I/O port, the program branches to the pre-defined line number.

(2) Format

```
IOMA
```

4.2.30 ON IO GOTO/ON IO GOSUB statements

(1) Function

These statements define the number of the line to be branched to when an interrupt is issued via an I / O port.

(2) Format

```
ON IO GOTO line number or *label
ON IO GOSUB line number or *label
```

- When there is an EVENT statement, it is executed before this statement.

4.2.31 Character size specification (DCHSIZE statement)

(1) Function

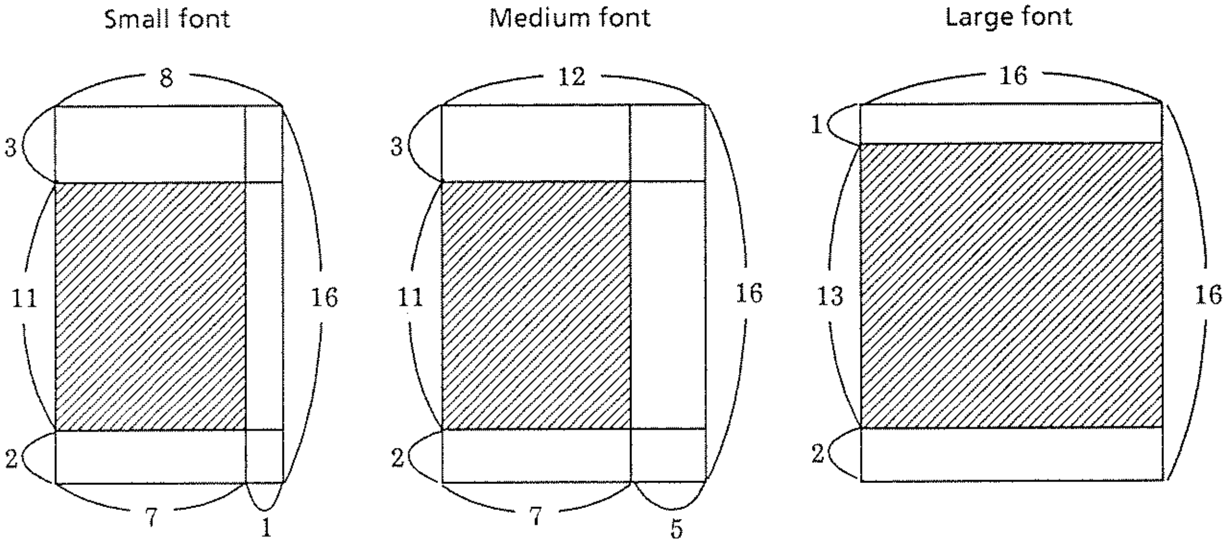
Specifies the display character size at system subroutine DCH execution.

(2) Format

```
DCHSIZE Character size number
```

Character size number	
0	Small font
1	Medium font
2	Large font

- The patterns of small / medium / large character fonts are shown below:



The units are dots on the CRT.

- The display character size can not be changed by PRINT statement, etc.
- Initialized by the RESET command.
- Small and medium font characters are the same size, but the character spacing is larger for medium font.

4.2.32 Home position (HOME statement)

(1) Function

This statement moves the cursor to the home position (upper left).

(2) Format

```
HOME
```

4.2.33 Delete (ERASE statement)

(1) Function

This statement deletes statements after the line with the cursor.

(2) Format

```
ERASE
```

Note: When only the middle intensity (PTA) screen is erased from the display, the screen is only partially erased. To erase the screen entirely, use the system subroutine CFL (see paragraph 5.2.2).

4.2.34 Time wait (WAIT statement)

(1) Function

This statement is used to wait for a specified time period.

(2) Format

```
WAIT { Numeric variable }
     { Numeric constant }
              |
       Waiting time (unit: second, 0.01 s resolution)
```

4.2.35 System subroutine execution (CALL statement)

(1) Function

This statement is used to execute system subroutines.
For details of system subroutines, see paragraph 5.2.

(2) Format

```
CALL system subroutine name [(parameter [,parameter...])]
```

4.2.36 ON ERROR statement

(1) Function

Registers the subroutine to branch (interrupt) to when an error occurs.

(2) Format

```
ON ERROR line number (or *label)
```

Notes:

- Execution is halted when an error occurs during the execution of an error processing subroutine.
- If there is an error statement right after the line where the error occurred, only the error statement will be executed.
- If the error is an execution termination error, no interrupt will occur.
- If an error occurs during data input with the INPUT statement, no interrupt will occur.
- The function ERRREAD (m) identifies the error code and line the error occurred. For details, see Paragraph 4.1.5 (5).
- Multiple interrupts with event interrupts are possible.
- The error occurred during an error interrupt processing is not applied.

4.2.37 OFF ERROR statement

(1) Function

Removes the registered subroutine to branch (interrupt) when an error occurs. No error interrupt will occur while after executing this command.

(2) Format

```
OFF ERROR
```

4.2.38 RETERR statement

(1) Function

Returns from an error interrupt.

Continues from the statement following the statement where the error occurred.

(2) Format

```
RETERR
```

Notes:

- If the RETURN or RETMAIN commands are used to return from an error interrupt ,an execution termination error (F243) will result.
- If the RETINT command is executed to return from an error interrupt, an execution termination error (F251) will result.
- If the RETERR command is executed when there is no error interrupt, an execution termination error (F252) will result.
- It is possible to branch to a normal subroutine (GOSUB··· RETURN) from the event interrupt subroutine.

4.2.39 RETRY statement

(1) Function

Returns from an error interrupt.

Execution is retried from the statement on which error occurred.

(2) Format

```
RETRY
```

Notes:

- If the RETURN or RETMAIN commands are used to return from an error interrupt, an execution termination error (F243) will result.
- If the RETINT command is executed to return from an error interrupt, an execution termination error (F251) will result.
- If the RETRY command is executed when there is no error interrupt, an execution termination error (F252) will result.
- It is possible to branch to a normal subroutine (GOSUB··· RETURN) from the event interrupt subroutine.

4.2.40 RESUME statement

(1) Function

Returns from an error interrupt.
Continues from the specified line.

(2) Format

```
RESUME line number (or *label)
```

Notes:

- If the RETURN or RETMAIN commands are used to return from an error interrupt, an execution termination error (F243) will result.
- If the RETINT command is executed to return from an error interrupt, an execution termination error (F251) will result.
- If a command other than the RESUME command is executed when there is no error interrupt, an execution termination error (F252) will result.
- It is possible to branch to a normal subroutine (GOSUB··· RETURN) from the event interrupt subroutine.

4.2.41 GIVEUP statement

(1) Function

Returns from an error interrupt.

Halts program execution.

(2) Format

```
GIVEUP
```

Notes:

- If the RETURN or RETMAIN commands are used to return from an error interrupt, an execution termination error (F243) will result.
- If the RETINT command is executed to return from an error interrupt, an execution termination error (F251) will result.
- If the GIVEUP is executed when there is no error interrupt, an execution termination error (F252) will result.
- It is possible to branch to a normal subroutine (GOSUB··· RETURN) from the event interrupt subroutine.

4.2.42 Error branch (ERROR statement)

(1) Function

To continue execution after warning-error generation, an ERROR statement can be used. Multiple lines can be used for ERROR statements. See paragraph 8.2 for details.

(2) Format

```
ERROR (error number,program line or *label to be executed
next)
```

4.2.43 Error main (ERRMAIN statement)

(1) Function

This statement branches to the highest level routine when an error that allows execution to continue (error code beginning with the letter W) is generated while the program was running.

(2) Format

```
ERRMAIN (error number)
```

Notes:

- When an ERRMAIN statement was executed in the highest level routine, the error code becomes F213.
- See paragraph 8.3 for details.

4.2.44 Data input 1 (READ statement)

(1) Function

This statement is used to receive data from a device connected to the GPIB or RS-232C through the specified port. When the specified port is a device port, the data is read from the dual port memory.

(2) Format

```
READ address,input variable[,input variable....]
READ address,variable[;]
```

- When ";" is not added at the end of the statement, commas (",") in the received data are assumed to be data delimiters and are stored in each variable.
- When ";" is added at the end of the statement, commas (",") are not assumed to be data delimiters and everything up to the data terminator is stored in one variable.

4.2.45 Data input 2 (BREAD statement)

(1) Function

This statement is used to receive one byte of binary data from a device connected to the GPIB or RS-232C through the specified port. When the specified port is a device port, this statement cannot be executed.

(2) Format

```
BREAD address,input variable[,input variable....]
```

- Data is not read from the dual port memory.

4.2.46 Data input 3 (WREAD statement)

(1) Function

This statement is used to receive one word of binary data from a device connected to the GPIB or RS-232C through the specified port. The data is stored in the input variable as high byte to low byte in sending order.
When the specified port is a device port, this statement cannot be executed.

(2) Format

```
WREAD address,input variable[,input variable....]
```

- Data is not read from the dual port memory.

4.2.47 Data output 1 (WRITE statement)

(1) Function

This statement sends data to a device connected to the GPIB or RS-232C through the specified port. When the specified port is a device port, data is written to dual port memory.

(2) Format

```
WRITE address,variable[:format][,variable[:format]...][;]
```

- The output data can also be a character constant.
- When ";" is added at the end of the statement, a terminator is not output.
- The output destination depends on the addressing method and GPIB1 port mode (system controller/device).

4.2.48 Data output 2 (BWRITE statement)

(1) Function

This statement sends one byte of binary data to a device connected to the GPIB or RS-232C through the specified port. When the specified port is a device port, this statement cannot be executed.

(2) Format

```
BWRITE address,variable[,variable ...]
```

Notes:

- Neither format nor ";" can be specified.
- The terminator is not output.
- Data is not written to the dual port memory.

4.2.49 Data output 3 (WWRITE statement)

(1) Function

This statement sends one word (two bytes) of binary data in order of high byte to low byte to a device connected to the GPIB or RS-232C through the specified port. When the specified port is a device port, this statement is not executed.

(2) Format

```
WWRITE address,variable[,variable...]
```

Notes:

- Neither format nor ";" can be specified.
- The terminator is not output.
- Data is not written to the dual port memory.
- When a one- or two-digit value is used (e.g. 5 or 17) for an address, the value becomes the address of the device connected to the port specified by the PORT command as a GPIB command (Indirect Port specification). However, when a three-digit value (e.g. 105 or 217) is used, the first digit becomes the port address and the lower two digits become the address of the device connected to the port (Direct Port specification).
- The lower two digits of the address at indirect or direct port specification have no meaning in the RS-232C. However, these digits should still be specified for form' s sake.

  Example:

  `WRITE␣5,"ABC"` .... Data is sent to address 5 through the port specified by the PORT command (indirect port specification).

  `WRITE␣1Ø5,"ABC"` ...... Data is sent to address 5 through port No. 1 (GPIB1) (direct port specification).

READ⌴217,A$ .... Data is input from address 17 through port No. 2 (GPIB2) (direct port specification).

READ⌴3ØØ,A$ .... Data entry from port No.3 (RS-232C) device (direct port specification)

These address specifications are effective for the WRITE, BWRITE, WWRITE, READ, BREAD, WREAD and LISTG statements.

The relationship between the port specification command and controller port is as follows:

	Indirect port specification	Direct port specification		
	WRITE 5	WRITE 1Ø5	WRITE 2Ø5	WRITE 3Ø5
At power-ON or after " PORT△1 " execution	*1 The GPIB1 port is a controller port.	*1 The GPIB1 port is a controller port.	The GPIB2 port is a controller port.	*2 The RS-232C port is the controller port.
After "PORT△2" execution	The GPIB2 port is a controller port.	*1 The GPIB1 port is a controller port.	The GPIB2 port is a controller port.	*2 The RS-232C port is the controller port.
After "PORT△3" execution	*2 The RS-232C port is the controller port.	*1 The GPIB1 port is the controller port.	The GPIB2 port is the controller port.	*2 The RS-232C port is the controller port.

*1: If the GPIB1 port is not designated as the controller port due to the CALL IFC statement, it will control the dual-port memory and then LISTG statement will become ineffective.

When the specified port is a device port, data is written to and read from the dual port memory. In this case, the BWRITE, WWRITE, BREAD, WREAD, and LISTG statements cannot be used.

*2: Addresses specified in the RS-232C have no meaning. However, these addresses should still be specified for form' s sake.

4.2.50 Data writing to the dual port memory (WDPM statement)

(1) Function

This statement writes data to the dual port memory.
See paragraph 6.5 for details.

(2) Format

```
WDPM memory number, variable[:format][, variable[:format]....]
```

Notes:

- The output data can also be character constants.
- ";" cannot be specified.
- This statement can be executed regardless of the GPIB1 mode (system controller/device).
- The dual port memory can also be written and read by the WRITE/READ statement. However, in this case, the statement cannot be executed if the GPIB1 port is not in device mode.

4.2.51 Data reading from the dual port memory (RDPM statement)

(1) Function

This statement reads data from the dual port memory.
See paragraph 6.5 for details.

(2) Format

```
RDPM memory number,input variable[,input variable ....]
```

- ";" cannot be specified.
- When data delimited by "," is input, multiple input variables are specified.

4.2.52 S.O.S (SOS)

(1) Function

This statement is displayed in the statement where a syntax error is generated during program loading.

(2) Format

```
SOS
```

Notes:

- A statement with SOS added is treated as a comment statement, the same as a REM statement, but when the program is run, it is treated as a syntax error.
- Line-number errors are treated as syntax errors (W6) and SOS is not displayed.

4.2.53 I/O port write strobe signal switching (OLDPORT statement)

(1) Function

Switches the generation timing of the write strobe pulse that is output when data is written to I/O ports C and D.

(2) Format

```
OLDPORT
```

- When this statement not executed
  A 1 μs write strobe signal pulse is generated about 1 μs after data is written. (Operation mode A)
- When this statement executed
  The write strobe signal pulse ends simultaneously with writing of data. (Operation mode B)

Notes:

- When latching the signal after data was written and stable data was output, use "operation mode A".
- When detecting the data change timing, use "operation mode B".
- When the power is turned on, "operation mode A" is the initial value.
- Once this statement has been executed, "operation mode B" is held until the power is turned on again.
- For a detailed description of the write strobe signal generation timing, see Section 9 I/O Port Control.

4.2.54 Setting the pseudorandom number sequence (RNDMIZE statement)

(1) Function

Sets a new initial value of a pseudorandom number sequence generated by the RND function.

(2) Format

```
RNDMIZE
```

Note: If this statement is not executed, the RND function in the program generates the same pseudorandom number sequence each time the program is executed.

SECTION 5
EXTENDED PTL

TABLE OF CONTENTS

(Blank)

There are system variables, system functions, and system subroutines in the extended PTL.

The extended PTL can execute high-speed operations, evaluate measurement results, and control external devices.

5.1 System Variables

PTA provides system variables with pre-defined names in addition to user-defined variables. These system variables can control I/O ports and read the measured data.

Variable name	Number of array elements	Purpose	Data meaning	Read/ Write
EX1	– – –	Corresponding to F1 key	Numbers 0 and 1 are switched alternately each time the F1 key is pressed.	R/W
EX2	– – –	Corresponding to F2 key	Numbers 0 and 1 are switched alternately each time the F2 key is pressed.	R/W
EX3	– – –	Corresponding to F3 key	Numbers 0 and 1 are switched alternately each time the F3 key is pressed.	R/W
EX4	– – –	Corresponding to F4 key	Numbers 0 and 1 are switched alternately each time the F4 key is pressed.	R/W
EX5	– – –	Corresponding to F5 key	Numbers 0 and 1 are switched alternately each time the F5 key is pressed.	R/W
EX6	– – –	Corresponding to etc key of each hierarchy	0 to 3: Switches a PTA function key hierarchy (*)	R/W
EXØ	– – –	I/O port control output	0 to 3: OUTPUT 1, OUTPUT 2 OUTPUT 1 OUTPUT 2 0 : OFF OFF 1 : ON OFF 2 : OFF ON 3 : ON ON	R/W
IOA	8	I/O port A control	0 to 255: Bit integers	R
IOB	8	I/O port B control	0 to 255: Bit integers	R
IOC	4	I/O port C control	0 to 15: Bit integers	R/W
IOD	4	I/O port D control	0 to 15: Bit integers	R/W
EIO	– – –	I/O port control (input-output switching)	0 to 3: IOC or IOD input-output switching PORT C PORT D 0 : Input Input 1 : Output Input 2 : Input Output 3 : Output Output	R/W

* Soft-key menus can be changed by inputting 0, 1, 2 and 3 to the system variable EX6, as shown below.

However, EX6 is disabled when the PTA menus are not being executed.

Variable name	Number of array elements	Purpose	Data meaning	Read/ Write
DTØ	– – –	Time setting/reading (year: Gregorian calendar)	0 to 99	R/W
DT1	– – –	Time setting/reading (month)	0 to 12	R/W
DT2	– – –	Time setting/reading (date)	0 to 31	R/W
DT3	– – –	Time setting/reading (hour)	0 to 23	R/W
DT4	– – –	Time setting/reading (minute)	0 to 59	R/W
XMA	1002	Waveform memory of TRACE-Freq	−32768 to 32767: 2-byte integer/1 point	R/W
XMB	1002	Occupied bandwidth and adjacent channel measurement (spectrum analyzer method)	−32768 to 32767: 2-byte integer/1 point	R/W
XMC	2×625	Constellation I/Q data	−32768 to 32767: 2-byte integer/1 point	R/W
XMD	4320	Power measurement	−32768 to 32767: 2-byte integer/1 point	R/W
XME	251	Occupied bandwidth measurement (FFT method)	−32768 to 32767: 2-byte integer/1 point	R/W
XMT	1002	Waveform memory of TRACE-Time	−32768 to 32767: 2-byte integer/1 point	R/W
SMA	1002	Submemory A	−32768 to 32767: 2-byte integer/1 point	R/W
SMB	1002	Submemory B	−32768 to 32767: 2-byte integer/1 point	R/W
IMA	1002	Image memory A	−32768 to 32767: 2-byte integer/1 point	R/W
IMB	1002	Image memory B	−32768 to 32767: 2-byte integer/1 point	R/W
RMA	1002	Real number memory A	8-byte floating point real number/1 point	R/W
RMB	1002	Real number memory B	8-byte floating point real number/1 point	R/W

	EX6 = 0	EX6 = 1	EX6 = 2	EX6 = 3
F1	RUN	PLIST	F1 *	YES
F2	STOP	CURSOR UP	F2 *	NO
F3	CONT	CURSOR DOWN	F3 *	(None)
F4	RESET	LOAD	F4 *	(None)
F5	PTA OFF	RUN	F5 *	(None)
F6	etc (1/4)	etc (2/4)	etc (3/4)	etc (4/4)

* Display characters can be defined with DEF subroutine.

5.2 System Subroutines

The MS8604A PTA has dedicated subroutines, called the system subroutines, executed by the CALL statement.

The system subroutines are shown below :

■ Display subroutines

● Displayed item erase :	`CALL CER(M)`
● Screen restore :	`CALL CRN(M)`
● Screen erase :	`CALL CFL(M)`
● Character-string display :	`CALL DCH(X,Y,text,M[,N])`
● Straight-line display :	`CALL DLN(XØ,YØ,X1,Y1,M[,N])`
● Square display :	`CALL DRC(XØ,YØ,X1,Y1,M[,N])`
● Circle display :	`CALL DCR(X,Y,R,M[,N])`
● Arc-line display :	`CALL DAR(XØ,YØ,RØ,W1,W2,M1[,M3])`
● Soft-key label registration :	`CALL DEF(M,text)`

■ Buzzer subroutine

● Buzzer :	`CALL BZR`

■ PMC subroutines

● File open (read) :	`CALL OPNI␣character string variable (or character constant)`
● File open (write) :	`CALL OPNO␣character string variable (or character constant)`
● File delete :	`CALL FDEL␣character string variable (or character constant)`
● Data load :	`CALL DALD variable`
● Data save :	`CALL DASV variable`
● File close :	`CALL CLS`
● Function save :	`CALL FNSV(M)`
● Function recall :	`CALL FNRC(M)`

■ GPIB subroutine (GPIB1 port only)

● Interface clear : (Changeover to system controller port)	`CALL IFC`
● Service request :	`CALL RSV(M)`
● Take controller :	`CALL TCT(M)`
● Changeover to device port :	`CALL DEV`

Interface subroutine

- Status byte reading : CALL GST (port number, address, input variable)
- Interface control : CALL GPIB (port number, control item number)

Panel subroutines

- Front-panel operation lock : CALL PNLL(Ø)
- Front-panel operation lock cancellation : CALL PNLU(Ø)

Waveform memory subroutine

- Memory copy : CALL COPY(MØ,M1)

Video plotter control subroutine

- Copy start : CALL VPT

Notes:

- If parameters specified in each subroutine are outside the specified range, an error occurs and no graphic data is plotted.
- The plotting subroutine (DCH, DLN, DRC, DCR, or DAR) cannot be used to plot graphic data on the ACRTC2 screen.

5.2.1 CER and CRN subroutines

(1) Function

The CER/CRN subroutines perform erasure and display restoration of the character string, graph, scale, marker, etc. on the CRT screen.

(2) Format

```
CALL␣CER(MØ) ....  Erases screen (MØ=Ø)
CALL␣CRN(MØ) ....  Restores items M0 display (MØ=Ø : display area except for waveform
                                              MØ=1 : waveform)
```

␣ indicates space

Notes:

- For the CRT screen details, refer to paragraph 1.4.
- A numeric constant or numeric variable is used for M0.
- When clear/display return was performed with this subroutine, the state is held until it is reset by this subroutine or until the PTA is turned off.
- In CRN displays, only the display area is displayed, the function key and menu title display areas are not displayed. (The PTA function key remains displayed in the function key display area. A PTA operating state remains displayed in the menu title display area.)
  A GPIB command whose operations include screen switching may be entered while the measuring instrument screen or a waveform is being displayed. In this case, the screen corresponding to the entered command is displayed.
- CER erases only the display area; it does not erase function key and menu title display areas.

5.2.2 CFL subroutine

(1) Function

This subroutine erases the two pairs (ACRTC1 and ACRTC2) in the four screens composed by brightness of high brightness, intermediate brightness, and low brightness, according to brightness.

(2) Format

```
CALL␣CFL(M1)
```

M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1
5	All screens of ACRTC1 and ACRTC2

Notes:

- A numeric constant or numeric variable is used for M1.
- This subroutine temporarily clears the screen.
  Therefore, when the display condition is established again, for example when the measurement parameters are changed or when characters are input from the PTA keyboard, they are displayed.
- See paragraph 1.4 for the screen details.

5.2.3 DCH subroutine

(1) Function

Displays a character string. (Referred to at the bottom left on the screen)

(2) Format

```
CALL␣DCH(X,Y,text,M1[,M2])
         | |   |    |  └── Character font specifications (normal/reversed display)
         | |   |    └───── Screen specification
         | |   └────────── Character string to be displayed
         | └────────────── Y coordinate of first character
         └──────────────── X coordinate of first character
```

M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1

M2	Display mode
0	Normal display
1	Reverse display

■ Range of each parameter

Font	First X coordinate (X)	First Y coordinate (Y)	Maximum No. of characters of string (text)
Small font	0 to 632	0 to 384	80
Medium font	0 to 628	0 to 384	60
Large font	0 to 624	0 to 384	40

Notes:

- The first X coordinate and Y coordinate specify the lower-left corner of the character.
- Numeric constants or numeric variables are used for X, Y, M1, and M2. "text" is a character constant or character variable.
- M2 is omissible and it is assumed to be 0 if omitted.
- The character size (small font/medium font/large font) can be set with the DCHSIZE statement.
  DCHSIZE 0: Small font
  DCHSIZE 1: Medium font
  DCHSIZE 2: Large font
- See paragraph 1.4 for the screen details.

5.2.4 DLN subroutine

(1) Function

This subroutine displays a straight line (sectional line).

(2) Format

```
CALL␣DLN (XØ,YØ,X1,Y1,M1[,M3])
```

Line type

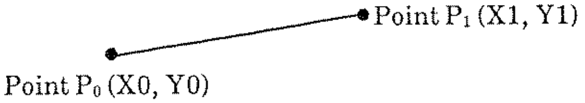
Screen type

Y coordinate at point P1 (0 to 399)

X coordinate at point P1 (0 to 639)

Y coordinate at point P0 (0 to 399)

X coordinate at point P0 (0 to 639)



M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1

M3	Line type
0	Displays solid line
1	Erases solid line
2	Displays dashed line
3	Erases dashed line

Notes:

- A numeric constant or numeric variable is used for X0, Y0, X1, Y1, M1, and M3.
- M3 is omissible and it is assumed to be 0 if omitted.
- See paragraph 1.4 for coordinate details.

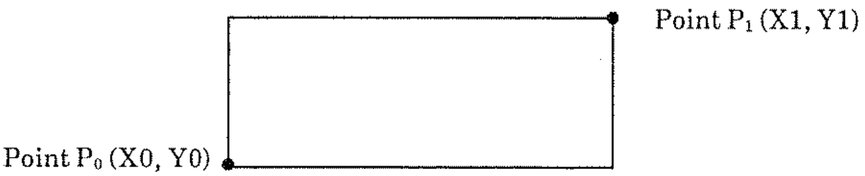
5.2.5 DRC subroutine

(1) Function

This subroutine displays a square based on a diagonal line between two specified points.

(2) Format

```
CALL␣DRC(XØ,YØ,X1,Y1,M1[,M3])
         |  |  |  |  |   └── Line type
         |  |  |  |  └── Screen type
         |  |  |  └──── Y coordinate at point P1 (0 to 399)
         |  |  └─────── X coordinate at point P1 (0 to 639)
         |  └────────── Y coordinate at point P0 (0 to 399)
         └───────────── X coordinate at point P0 (0 to 639)
```



M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1

M3	Line type
0	Displays solid line
1	Erases solid line
2	Displays dashed line
3	Erases dashed line

Notes:

- A numeric constant or numeric variable is used for X0, Y0, X1, Y1, M1, and M3.
- M3 is omissible and it is assumed to be 0 if omitted.
- See paragraph 1.4 for coordinate details.
- No display is performed if P0 (X0, Y0) and P1 (X1, Y1) are at the same axis.

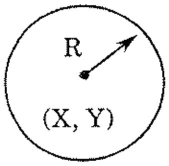
5.2.6 DCR subroutine

(1) Function

This subroutine displays a circle.

(2) Format

```
CALL␣DCR(X,Y,R,M1[,M3])
         | | | |    └──── Line type
         | | | └── Screen type
         | | └──── Radius (1 to 1506)
         | └────── Y coordinate of center (−399 to 798)
         └──────── X coordinate of center (−639 to 1278)
```



M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1

M3	Line type
0	Displays solid line
1	Erases solid line
2	Displays dashed line
3	Erases dashed line

Notes:

- Numeric constants or numeric variables are used for X, Y, R, M1, and M3.
- M3 is omissible and it is assumed to be 0 if omitted.
- See paragraph 1.4 for coordinate details.

5.2.7 DAR subroutine

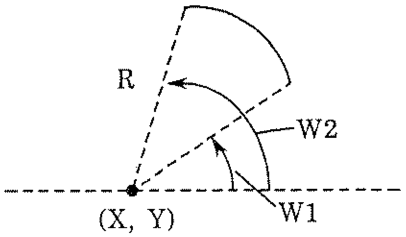
(1) Function

Displays an arc.

(2) Format

```
CALL␣DAR(X,Y,R,W1,W2,M1[,M3])
```

- M3: Line type
- M1: Screen type
- W2: Plot end angle (−180.00 to 180.00 deg.)
- W1: Plot start angle (−180.00 to 180.00 deg.)
- R: Radius (1 to 1506)
- Y: Coordinate Y of center (−399 to 798)
- X: Coordinate X of center (−639 to 1278)



M1	Screens
0	All screens of ACRTC1
1	Intermediate brightness 1 screen of ACRTC1
2	Low brightness screen of ACRTC1
3	Intermediate brightness 2 screen of ACRTC1
4	High brightness screen of ACRTC1

M3	Line type
0	Displays solid line
1	Erases solid line
2	Displays dashed line
3	Erases dashed line

Notes:

- Numeric constants or numeric variables are used for the X, Y, R, W1, W2, M1, and M3.
- M3 is omissible and it is assumed to be 0 if omitted.
- See paragraph 1.4 for coordinate details.

5.2.8 DEF subroutine

(1) Function

Registers a menu label (name) in the soft key menu.

When the PTA menu (3/4) is displayed, the labels registered by this subroutine are displayed.

(2) Format

```
CALL␣DEF(M,text)
          |  └─── Name of 30 characters maximum
          |
          └────── Soft-key number (1 to 6)
```

Notes:

- M is a numeric constant or numeric variable.
- "text" is a character constant or character variable.
- The labels registered by this subroutine remain valid until the PTA is turned off.

5.2.9 BZR subroutine

(1) Function

This subroutine sounds the buzzer once.

(2) Format

```
CALL␣BZR
```

Note: If the buzzer was disabled with the GPIB command, the buzzer does not sound even if this subroutine is executed.

5.2.10 OPNI, OPNO and FDEL subroutines

(1) Function

Opens a data file to write data to and read data from a PMC or FD and deletes an existing data file.

(2) Format

```
CALL␣OPNI␣character string-variable (or character constant)
Open data read
CALL␣OPNO␣character string-variable (or character constant)
Open data write
CALL␣FDEL␣character string-variable (or character constant)
Delete data file
```

Notes:

- The data file name always begins with a % symbol and is followed by 6 or less alphanumeric characters including %.
- Do not remove the PMC or FD while opening the data file in it.
- This subroutine cannot be used with PTA program files.

5.2.11 DALD and DASV subroutines

(1) Function

The DALD subroutine reads the data saved in the Plug-in Memory Card (PMC) or FD ; the DASV subroutine saves the data in the PMC or FD.

(2) Format

```
CALL␣DALD␣input variable: Read data from data file
CALL␣DASV␣variable      : Write data to data file
```

Notes:

- Data files are created as sequential files.
  Therefore, read them in the order in which they were written.
- Different types of data (for example, numeric type and character type) can be stored in one data file. However, when the type when the data was written and the type of input variable when the data was read cannot be assigned, an error is generated.

(3) Program example

```
 10 REM "*** DATA FILE ***"
 20 CALL OPNO"%DATA"
 30 FOR C=2 TO 10
 40 D=C*C                            Writing into file
 50 CALL DASV D
 60 NEXT C
 70 CALL CLS
 80 CALL OPNI"%DATA"
 90 FOR C=2 TO 10
100 CALL DALD D
110 PRINT "RES.=",D                  Reading from file
120 NEXT C
130 CALL CLS
140 STOP
```

(4) Executed results

```
RES.=4
RES.=9
RES.=16
RES.=25
RES.=36
RES.=49
RES.=64
RES.=81
RES.=100
```

5.2.12 CLS subroutine

(1) Function

This subroutine closes the open data file.
Used for both write and read.

(2) Format

```
CALL␣CLS
```

5.2.13 FNSV and FNRC subroutine

(1) Function

This subroutine saves (FNSV) the measurement screen into the internal memory (1 to 4) or recalls (FNRC) from the internal memory.

(2) Format

```
CALL␣FNSV(M)
CALL␣FNRC(M)
            |
       Function memory No. (1 to 4)
```

Notes:

- M is constant or variable.
- The function of this subroutine is the same as the saving or recalling to the internal memory by ordinary key operation.
- When saved or recalled into the PMC or FD, use "SVM/RCM" of the GPIB command by the PUT statement. See paragraph 4.2.20 for details.

5.2.14 IFC subroutine

(1) Function

When this subroutine is executed, the GPIB1 port becomes the system controller and outputs the “Interface clear ” signal to the devices connected to the GPIB1.

(2) Format

```
CALL␣IFC
```

Notes:

- The GPIB2 port always operates in system controller mode.
- When the PTA is turned off, the GPIB1 port reverts to device port.

5.2.15 RSV subroutine

(1) Function

This subroutine sends the service request to the controller when the GPIB1 port (the first interface) is used as a device port.

(2) Format

```
CALL␣RSV(M)
```

M	PTA Event Status Register							
	MSB	- - -	- - -	- - -	- - -	- - -	- - -	LSB
0	×	×	×	×	0	0	0	1
1	×	×	×	×	0	0	1	0
2	×	×	×	×	0	0	1	1
3	×	×	×	×	0	1	0	0
4	×	×	×	×	0	1	0	1
5	×	×	×	×	0	1	1	0
6	×	×	×	×	0	1	1	1
7	×	×	×	×	1	0	0	0
8	×	×	×	×	1	0	0	1
9	×	×	×	×	1	0	1	0

(× means don't-care bit which does not change.)

The PTA event status register is defined as the extended status of Status-Byte bit 1.

Therefore, setting the left-described data (into the PTA Even Status Register) indirectly sets Status-Byte bit 1 as a summary bit.

The RQS bit (bit 6) is set as the logical AND of each Status-Byte bits to issue a service request to the controller.

The GPIB commands (used to read the Status Byte and PTA Event Status Register from the external controller) are *STB ? and ESR1 ?, respectively.

Notes:

- A numeric constant or numeric variable is used for M.
- This subroutine is effective only when the GPIB1 port is a device port.

5.2.16 TCT subroutine

(1) Function

This subroutine causes controlling right to be passed to another device provided that the GPIB1 port is used as a system controller port.

(2) Format

```
CALL␣TCT(M)
         |
Address of device to which control right is passed.
```

Notes:

- M is the GPIB address from 0 to 30, and a numeric constant or numeric variable is used.
- This subroutine is effective only when the GPIB1 port is a system controller port.

5.2.17 DEV subroutine

(1) Function

This subroutine causes the GPIB1 port to become a device port when it has previously been used as the system controller.

(2) Format

```
CALL␣DEV
```

- The GPIB2 port is always a system controller port.
- This subroutine is effective only when the GPIB1 port is a system controller port.

5.2.18 GST subroutine (GST)

(1) Function

A serial poll is executed on the device specified by address when the specified port is for the controller. The status value is read and stored in an input variable.

(2) Format

```
CALL␣GST(P, address, input variable)
         |    |           └─ Numeric variable
         |    └───────────── GPIB address on the device (0 to 30)
         └────────────────── Specified port (1: GPIB1, 2: GPIB2)
```

- The read status value will be stored in the input variable. Input variable can be either a real-number, integer, or bit type variable.
- This subroutine is effective only when the specified port is a system controller port.
- This subroutine cannot be executed on the RS-232C.

5.2.19 Interface control subroutine (GPIB and RS-232C)

(1) Function

The "Interface Clear" (IFC), "Remote" (REN), "Local" (LCL), "Device Clear" (DCL), "Local Rockout" (LLO), and "Device Trigger" (DTR) are sent, and "Return to Local" (RTL) is set from the specified port.

(2) Format

`CALL␣GPIB(P,Ø)`	Sends IFC .
`CALL␣GPIB(P,1[, address])`	Sends REN .
`CALL␣GPIB(P,2)`	Sends RTL .
`CALL␣GPIB(P,3[,address])`	Sends LCL .
`CALL␣GPIB(P,4[,address])`	Sends DCL .
`CALL␣GPIB(P,5)`	Sends LLO .
`CALL␣GPIB(P,6,address)`	Sends DTR .

P : Specified port No. (GPIB1:1,GPIB2:2, RS-232C:3)

Address : GPIB device address of 0 to 30

Notes:

- P and address are numeric constants or numeric variables.
- The actions of each subroutine are described below.

IFC:

- The IFC line is turned on for 100 μsec. The interface functions of all connected devices are initialized.
- Initialization is executed only for the corresponding interface functions. This code does not affect device functions.
- All talkers and listeners are not released.
- This does not affect the SRQ line.
- If the system passes control of the GPIB1 port to other controllers with the CALL TCT (m) command, control will be automatically returned to the PTA when execution is finished.
- This subroutine terminates normally without performing any processing for the RS-232C.

REN:

- When [, address] is omitted, the REN line is turned ON. Afterwards when the device is set to listener, it will assume remote control status.
- When [, address] is specified, the REN line is turned on. The device specified by [, address] will be identified as the listener and assume remote control status.
- Can be executed only when the specified port is a system controller port.
- This subroutine terminates normally without performing any processing for the RS-232C.

RTL:

- When the GPIB1 port is identified as the device, the PTA assumes the local control status. (This has the same effect as pressing the [LOCAL] key.)
- Only "1" can be specified as the port No.

Notes: (Continued)

LCL:
- When [, address] is omitted, the REN line is turned off. All devices assume local control status.
- When [, address] is specified, all listeners are released. After that, the device specified by [, address] is selected as the listener and assumes local control status. The REN line does not change.
- Can be executed only when the specified port is a system controller port.
- This subroutine outputs the "GTL" + <CR+LF> command for RS-232C.

DCL:
- When [, address] is omitted, "DCL" is sent and all device functions on the GPIB are initialized.
- When [, address] is specified, (Selected Device Clear) is sent and the device function specified by [, address] is initialized.
- Can be executed only when the specified port is a system controller port.
- This subroutine outputs the "DCL" + <CR+LF> command for RS-232C.

LLO:
- Disables the remote to local switching function of all devices on the GPIB. You will not be able to switch the device to local with the [Local] key on the panel.
- Switching is possible with the REN and LCL commands from the PTA.
- This mode can be exited with the LCL command in which the [, address] is omitted.
- Can be executed only when the specified port is a system controller port.
- This subroutine outputs the "LLO" + <CR+LF> command for RS-232C.

DTR:
- Triggers the specified device. The specified device begins the predetermined operation.
- Can be executed only when the specified port is a system controller port.
- This subroutine terminates normally without performing any processing for the RS-232C.

5.2.20 PNLU and PNLL subroutine

(1) Function

Sets LOCK/UNLOCK on the front panel when PTA is on.

(2) Format

```
CALL␣PNLU(Ø)
CALL␣PNLL(Ø)
```

Note: The front-panel soft keys [F1] to [F6], [Shift], and [Local] cannot be lock-out.

5.2.21 COPY subroutine

(1) Function

This subroutine copies the data in a specified waveform memory (copy source) to another waveform memory (copy destination). For example, use of the sub memory permits measurement in parallel with data processing.

(2) Format

```
CALL␣COPY (MØ, M1)
           |    |
           |    └──── Destination memory
           |
           └───────── Source memory
```

M0, M1	Memory	System variable name	Type
0	Measurement memory	XMA	Integer (0.01 dBm unit)
1	Measurement memory	XMB	Integer (0.01 dBm unit)
2	Submemory a	SMA	Integer (0.01 dBm unit)
3	Submemory b	SMB	Integer (0.01 dBm unit)
4	Image memory a	IMA	Integer
5	Image memory b	IMB	Integer
6	Real number memory a	RMA	Real number
7	Real number memory b	RMB	Real number
8	Measurement memory	XMT	Integer

Notes:

- M0 contents are copied in M1. M0 contents are not changed. Previous contents of M1 are lost.
- A numeric constant or numeric variable is used for M0 and M1.
- Data cannot be copied between integer memory and real number memory.

5.2.22 VPT subroutine

(1) Function

Controls the start of hard copying by the UA455A (Video Plotter).

(2) Format

```
CALL␣VPT
```

Notes:

- This subroutine is effective when separate video signals (round DIN connector) are supplied to the UA455A (Video Plotter) from the MS8604A.
- The UA455A is remotely controlled.
- The UA455A can be controlled regardless of the MS8604A hard copy parameter settings.

SECTION 6
GPIB IN PTA

TABLE OF CONTENTS

(Blank)

6.1 General

The MS8604A is equipped with two GPIB ports. The GPIB1 port mainly enables the MS8604A to be remotely controlled by external host computer. The GPIB2 port mainly controls external equipment from the PTA. Therefore, using the GPIB1 port as the device port and the GPIB2 port as the system controller port enables effective system configuration.

6.1.1 GPIB1/GPIB2 port functions

(1) GPIB1 function

The GPIB1 port can be operated as the same procedure as that of a conversional instrument equipped with a one-port GPIB. Therefore, at power-on or in the normal measuring state, GPIB1 functions as a device port. By setting a system controller, the GPIB1 functions as a system controller port in the PTA mode to control other devices. For a small system, the conversional PTA programming environment can be used as is without considering the GPIB2 port.

(2) GPIB2 function

Independent of the GPIB1 port, the GPIB2 is used as the control port for the devices connected to the GPIB2 port. Therefore, the GPIB2 always functions as a system controller port, and cannot be used as a device port.

6.2 Setting GPIB1 Port to the System Controller

The PTA is turned on when the [PTA] key is pressed. When the IFC subroutine is executed (CALL IFC) by the program or through immediate execution, the GPIB1 port is set as the system controller. In addition, when the DEV subroutine (CALL DEV) is executed, GPIB1 returns to a device port Further, when the PTA is turned off, GPIB returns to the device port. The GPIB2 port is only set in the system controller state.

6.3 Device Messages in PTA

(1) Device message from external controller

The GPIB device messages in PTA are shown below. These device messages are messages which are used to control the MS8604A from an external controller when the PTA GPIB1 port is not set as the controller.

Function		Device message	
PTA execution status	Control	PTA␣1	; Turns the PTA on (execution status).
		PTA␣Ø	; Turns the PTA off.
	Request	PTA?	; Requests the execution status of the PTA. Response 0 : READY (program execution stop state) 1 : BREAK (program suspension state) 2 : BUSY (immediate-execution-command execution state) 3 : RUN (program execution state)
PTL MODE	Control	PTL␣1	; Sets the PTL mode to ON. When in PTL mode, the data input from an external controller is assumed to be PTL command to process commands and programs.
		PTL␣Ø	; Sets the PTL mode to OFF.
	Request	PTL␣2	; Outputs the program to the external controller from the next reading
Dual-port memory	Control	PMY␣m1,m3	; Specifies memory number m3: Data m1: Memory number 0 to 31
	Request	PMY?␣m1,m2	; Requests memory data m2: Number of requested memories 1 to 32. m1: Starting memory number to be requested 0 to 31.

Note: The PTL mode is turned OFF (PTL 0) after CALL IFC, CALL RSV (M) or CALL GPIB (1, n) of the GPIB1 interface function command is executed.

(2) PTA dedicated GPIB commands (Used by WRITE 1000, READ 1000)

When setting or reading parameters with PTA, use the WRITE 1000 or READ 1000 statement and send GPIB format device messages to the main frame MS8604A.

The following messages can be sent by the PTA besides the GPIB commands in the MS8604A.

Function		Message		
Port Switching	Control	`PORT␣1`	;	Selects GPIB1 as the PTA control port.
		`PORT␣2`	;	Selects GPIB2 as the PTA control port.
		`PORT␣3`	;	Selects the RS-232C as the PTA controller port.
	Request	`PORT?`	;	Requests the PTA control port.
Event Occurrence DELAY (Clock 1)	Control	`EDLY␣t`	;	Sets the DELAY time an event interrupt will occur. DELAY time: 1 seconds up to 1 hour (in 1 s step)
Event Occurrence TIME (Clock 2)	Control	`ETIM␣t1,t2,t3`	;	Sets the time an event interrupt will occur. Seconds: Up to 59 seconds Minutes: Up to 59 minutes Hours: Up to 23 hours
Event Occurrence CYCLE (Clock 3)	Control	`ECYC␣t`	;	Sets the cycles an event interrupt will occur. Cycle: 1 seconds up to 1 hour (in 0.1 s steps)

Notes:

- For details on the WRITE 1000 and READ 1000 statements, see paragraphs 4.2.20 and 4.2.21.
- For details on event interrupts, see paragraph 4.2.23.
- The control port (for the WRITE, READ, LISTG statements and other GPIB statements supported by the PTA) is the port selected by the PORT command except when these statements are executed with a direct port specification.
  In the initial state, the GPIB1 port is selected as the PTA control port.
- Ports specified by the port switching command are not initialized by PTA → OFF. Port 1 (GPIB1 port) is initialized only when the power is turned on.

(3) Other GPIB commands (MS8604A-specific commands)

Even if PTA is on, measuring instrument-specific commands are accepted (PTL mode is off).

Measurement values or waveforms are not displayed on PTA screens. When the measuring instrument screen is displayed by selecting the display item, measurement values and waveforms are displayed and updated. A command whose operations include screen switching may be entered while the measuring instrument screen is being displayed. In this case, the screen corresponding to the entered command is displayed (see Paragraph 5.2.1).

(4) GPIB device messages of PTA

Function name	Item name	MS8604A			Remarks
		Program Msg	Query Msg	Response Msg	
PTA					
PTA Sw	On	PTA 1	PTA?	PTA n Ø: ready, 1: break 2: busy 3: run	
	Off	PTA Ø	PTA?	Ø	
PTL mode	ON	PTL Ø	——	——	
	OFF	PTL 1	——	——	
	OUT	PTL 2	——	…………	Program output
GPIB 2 address	address	GPIA n	GPIA?	GPIA n	Even if PTA is off, this function is valid.
Dual Port memory		PMY i, "text"	PMY? i, d	b………	
		The text is the character string enclosed in double or single quotation marks.		<Up to 32 characters>	
Control port select	GPIB1	PORT 1	PORT?	PORT 1	
	GPIB2	PORT 2	PORT?	PORT 2	
	RS-232C	PORT 3	PORT?	PORT 3	
EVENT occurrence condition	delay	EDLY m	——	——	
	Hour	ETIM m1, m2, m3	——	——	
	Cycle	ECYC m	——	——	
<System subroutine>					
Erase		CER Ø	——	——	
Restore	Other than waveform	CRN Ø	——	——	
	Waveform	CRN 1	——	——	
Save to memory	Memory 1	RGSV 1	——	——	
	Memory 2	RGSV 2	——	——	
	Memory 3	RGSV 3	——	——	
	Memory 4	RGSV 4	——	——	
Recall from memory	Memory 1	RGRC 1			
	Memory 2	RGRC 2	——	——	
	Memory 3	RGRC 3	——	——	
	Memory 4	RGRC 4	——	——	
Video-Plotter copy start	UA455A	VPT	——	——	
Panel unlock		PNLU Ø	——	——	
Panel lock		PNLL Ø	——	——	

6.4 Function as Controller/Device

6.4.1 Function as controller

(1) Program listing

Lists programs to an external printer by using the LISTG command through the current GPIB port.

(2) IFC sending

Sends the "Interface Clear" to the device on the GPIB1 by using the CALL⎵IFC statement. The GPIB1 port returns to the device port by turning off the PTA.

(3) Controller right allocation

Allocates controller right to the device with the address specified by M by using the CALL⎵TCT (M) statement .

(4) Data sending

Sends the data to the device on the GPIB by using the WRITE statement

```
WRITE⎵M,Variable[:Format][,Variable[:Format]····]
      |            |
      |            Output data (A character constant is possible.)
      Address of external device (A numeric constant or numeric variable is used.)
```

Note: When M is 1000, the functions of the MS8604A main frame are set. Also, this operations are performed in either the controller or device mode at this time.

(5) Data reception

Receives the data from the device on the GPIB by using the READ statement

```
READ⎵M,Variable [,Variable·····]
     |    |
     |    Received data is input in variable.
     Address of external device (A numeric constant or numeric variable is used.)
```

Note: When the specified GPIB port is the device port, WRITE and READ statements access the dual-port memory.

Note: When one- or two-digit value (e.g.,5 or 17) is specified for an address, the value indicates the address of the device connected to the port specified by the PORT command of the GPIB command (Indirect Port Specification). When a three-digit value (e.g.,105 or 217) is specified, the high-order digit indicates the port number, and two low-order digits indicate the address of the device connected to the port indicated by the above port number. (Direct Port Specification). The two lower digits of an address at indirect or direct port specification have no meaning in RS-232C. However, these digits should still be specified for form's sake.

Example:

`WRITE␣5,"ABC"` .... Data is sent to address 5 through the current port (indirect port specification).

`WRITE␣1Ø5,"ABC"` .... Data is sent to address 5 through the specified port No.1 (GPIB1) (direct port specification).

`READ␣217,A$` .... Data is input from address 17 through the specified port No.2 (GPIB2) (direct port specification).

`READ␣3ØØ,A$` .... Data entry from port No.3 (RS-232C) device (direct port specification)

These address specifications are effective for the WRITE, BWRITE, WWRITE, READ, BREAD, WREAD and LISTG statements.

The relationship (between the port specification command and controller port) is as follows:

	Indirect port specification	Direct port specification		
Condition	WRITE 5	WRITE 1Ø5	WRITE 2Ø5	WRITE 3Ø5
At power-on or after "PORT␣1" execution	*1 The GPIB1 port is the controller port.	*1 The GPIB1 port is the controller port.	The GPIB2 port is the controller port.	*2 The RS-232C port is the controller port.
After "PORT␣2" execution	The GPIB2 port is the controller port.	*1 The GPIB1 port is the controller port.	The GPIB2 port is the controller port.	*2 The RS-232C port is the controller port.
After execution of "PORT␣3"	*2 The RS-232C port is the controller port.	*1 The GPIB1 port is the controller port.	The GPIB2 port is the controller port.	*2 The RS-232C port is the controller port.

*1 If the GPIB1 port is not the controller port due to the CALL IFC statement, it controls the dual port memory. In this case, the LISTG statement becomes ineffective.

When the specified port is a device port, data is written to and read from the dual port memory. In this case, the BWRITE, WWRITE, BREAD, WREAD, and LISTG statements cannot be used.

*2 Address specification in the RS-232C has no meaning. However, the address should still be specified for form's sake.

(6) Time out

Time-out value is 5 s (initial value) for both the GPIB1 and GPIB2 ports. (This value is applied only at the system controller state for the GPIB1 port.)

The following GPIB command is used for change of thime-out value.

```
GTOUT␣t      t=0 to 225 s (in 1 s steps)
```

When t=0 is specified, no time-out is set.

(7) Terminating Codes for READ/WRITE Statements

The following terminating codes are used for the GPIB1 and GPIB2 ports.

Talker (send) terminators

<Port> command	Terminator code
<GPIB1> WRITE LISTG	Depends on TRM command. eiter CR+LF or LF
<GPIB2> WRITE LISTG	CR+LF (Fixed)

Note:
The TRM command shown below is a GPIB command.

```
TRM␣1    (CR+LF)
TRM␣Ø    (LF only)
     Initial value :Ø
```

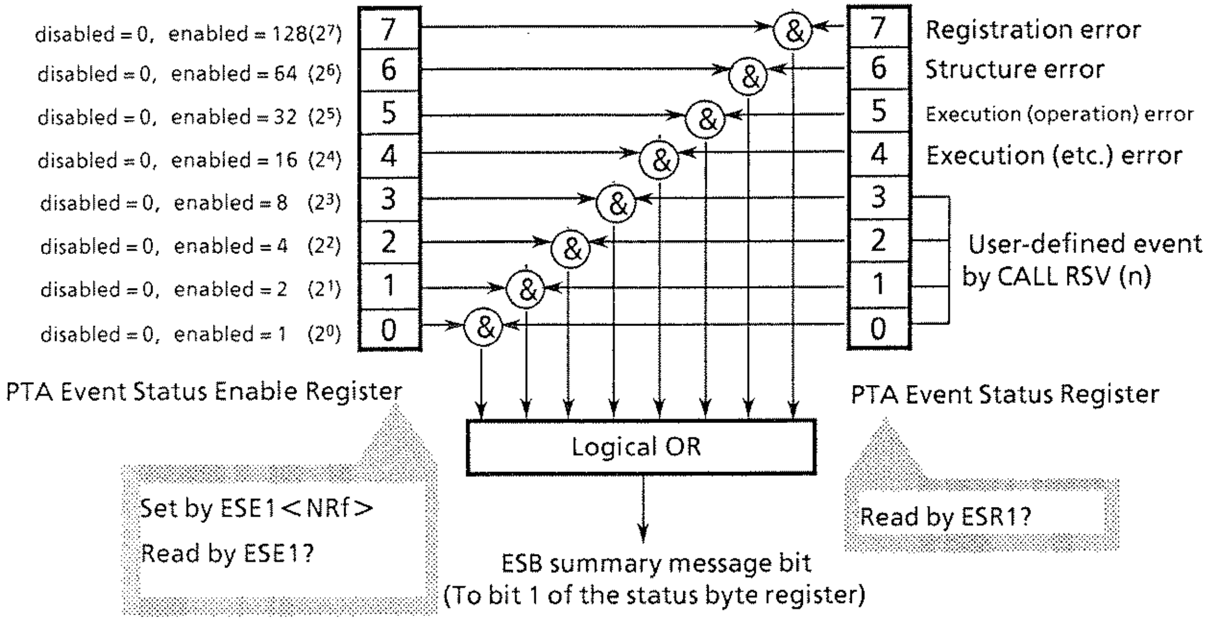
Listener (receive) terminators

<Port> command	Terminator code
<GPIB1> READ	LF or CR+LF
<GPIB2> READ	LF or CR+LF

6.4.2 Function as device

(1) Service request sending

Sends a service request command to an external controller by using the CALL␣RSV (M) statement.

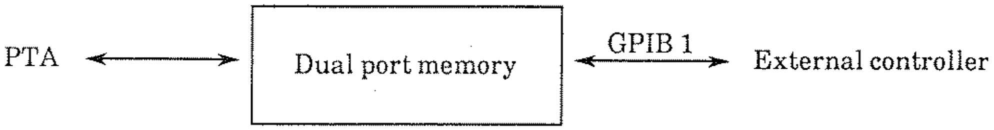

disabled = 0, enabled = 128(2⁷)
disabled = 0, enabled = 64 (2⁶)
disabled = 0, enabled = 32 (2⁵)
disabled = 0, enabled = 16 (2⁴)
disabled = 0, enabled = 8 (2³)
disabled = 0, enabled = 4 (2²)
disabled = 0, enabled = 2 (2¹)
disabled = 0, enabled = 1 (2⁰)
7
6
5
4
3
2
1
0
&
Registration error
Structure error
Execution (operation) error
Execution (etc.) error
User-defined event
by CALL RSV (n)
PTA Event Status Enable Register
PTA Event Status Register
Logical OR
Set by ESE1<NRf>
Read by ESE1?
Read by ESR1?
ESB summary message bit
(To bit 1 of the status byte register)

Bit	Event name	Description
7	Registration error	Error at program registration
6	Structure error	Error on program structure
5	Execution (operation) error	Error at operation on program execution
4	Execution (etc.) error	Error at other than program operation
3	(User-defined event)	(User defined by CALL RSV (n))
2	(User-defined event)	(User defined by CALL RSV (n))
1	(User-defined event)	(User defined by CALL REV (n))
0	(User-defined event)	(User defined by CALL RSV (n))

6.5 Dual Port Memory

(1) Application and configuration

The dual port memory is built into the PTA. Data can be freely written and read from the PTA and from an external GPIB.
The data and measured result obtained in the PTA program are output to an external controller via this memory and are used to communicate between the PTA and the external controller.
The external controller writes and reads the dual port memory through the GPIB1 port.



The dual port memory consists of thirty-two 32-byte memories. The memories are accessed by specifying the memory number.
Memory numbers from 0 to 31 can be specified.

Dual port memory configuration

Memory No. 0	32 bytes
Memory No. 1	32 bytes
Memory No. 2	32 bytes
⋮	⋮
Memory No. 30	32 bytes
Memory No. 31	32 bytes

(2) Writing data to dual port memory

Format

- Writing from PTA

```
WDPM memory number, write data  or
WRITE memory number write data  or
PUT (or WRITE 1000) " PMY memory number, write data"
```

- Writing from external controller

```
" PMY memory number, write data"
```

- When writing data to the dual port memory, be sure to specify the memory number. Data is written sequentially, beginning from the first byte of the specified memory number.
- A 1-byte termination code (LF) is added at the end of the write data.
- When the write data size exceeds 32 bytes, it can be written to the next memory. When the write data size is exactly 32 bytes, the termination code is stored at the beginning of the next memory. However, when data has been written up to the last byte of the last memory number, the termination code is not added.
- When writing past the last byte of the last memory number is attempted, an error is generated and writing is not performed. In this case, the previously written data is retained.
- Data is always stored in memory as ASCII data. When data is written from the PTA, its storage size differs as follows, depending on the type of data:

① Character constant/variable

- Written as 1 byte/1 character ASCII data.
- When unformatted character variable data is written, (number of bytes of array size) + (1 byte: space code) is written. The termination code is written at the end.
- When upper case formatted character variables are used, a 1-byte space code is written at the end of the data. The termination code is written at the end.
- When character variables are used, the number of characters in "   " are written. The termination code is written at the end.

② Numeric variable

- Numerics are converted to character strings (ASCII data) and data of that size is written.
  The minus sign and decimal point require one byte each.
  The termination code is written last.

③ Bit variable

- The 0/1 numeric of each bit is converted to a character string (ASCII data) and data of that size is written as 1 byte/1 bit.
- The storage format when the data is formatted/unformatted is the same as when character variables are used.
- Writing from the PTA with the WRITE statement is effective only when the GPIB1 port is a device port.
- The BWRITE and WWRITE statements cannot be used.

Examples:

- Writing from PTA

```
WDPM Ø, "MEASEND"    : Write "MEASEND" to Memory No. 0.
WRITE 1, RES         : Write numeric variable RES data to Memory No. 1.
```

- Writing from external controller

```
"PMY Ø, MEASSTART"   : Write "MEASSTART" to memory No.0.
```

Notes:

- The WDPM statement is a dedicated statement for writing data to dual port memory.
- The WRITE statement is a general-purpose output statement. Output to GPIB is also possible. The output destination depends on whether the specified controller is a system controller or device state.
- The PUT or WRITE 1000 statement is mainly used to set the MS8604A measurement parameters. However, messages in the same format as setting from an external controller can be written by using these statements to send GPIB format messages from the PTA.

(3) Reading data from dual port memory

Format

- Reading from PTA

  RDPM memory number, input variable [, input variable..] or
  READ memory number, input variable [, input variable..] or
  PUT (or WRITE 1000) " PMY? read start memory number, number of memories"
  +READ 1000, input variable [, input variable]

- Reading from external controller

  " PMY? read start memory number, number of memories" + read command

---

- When reading data from the dual port memory, be sure to specify the memory number. Everything up to the termination code (LF) is, as a rule, output as one data item.
  However, when dual port memory was read up to the last byte of the last memory number, the data is assumed to end at that point.
- When data was written over multiple memories and is read by specifying an intermediate memory number, the intermediate data is read.
- As a rule, when data is read from the PTA, the data up to the termination code is read. However, if the data contains commas (" , "), the commas are assumed to be delimiters and the data up to the front of the comma is stored in the input variable. Therefore, in this case, multiple input variables must be specified.
  When the number of delimited data and the number of input variables is different, a write error (when the number of input variables is large) may be generated, or the output data may remain inside (when the number of input variables is small).
  To avoid a comma being considered a data delimiter, store the data up to the termination code in one input variable by specifying " ; " at the end of the statement.
  In this case, only one input variable can be specified.
- When data is read from an external controller and when data is read from the PTA with the PUT or WRITE 1000 statement, use the "PMY?" command. The "PMY?" command can specify the read start memory number and the number of memories to be read. In this case, the data from the beginning to the termination code of each memory number is delimited into the specified number of memories by commas and is output.
- When the data in the dual port memory is assigned to input variables, it may not be possible to assign the data to an input variable type different from the assignment data. In this case, a read error is generated.
- Data can be read by READ statement only when the GPIB1 port is a device port.
- The BREAD and WREAD statements cannot be used.

Examples:

- Reading from PTA

  `RDPM Ø, A$` : Read data from Memory No. 0 and store it in character variable A$.

  `READ 1, B` : Read data from Memory No. 1 and assign it to numeric variable B.

- Reading from external controller

  `"PMY? Ø, 3"` : Issue a memory data output request for Nos. 0 to 3 (memory Nos. 0, 1, 2).

Notes:

- The RDPM is a dedicated statement for reading data from dual port memory.
- The READ statement is a general-purpose output statement. Input from GPIB is also possible. The input destination depends on whether the specified port is a controller or device state.

(4) Details of write/read the dual-port memory

Control command from external controller

```
"PMY␣Ø,ABC"
"PMY␣1,123"
"PMY␣2,XYZ"
```

Contents of dual-port Memory

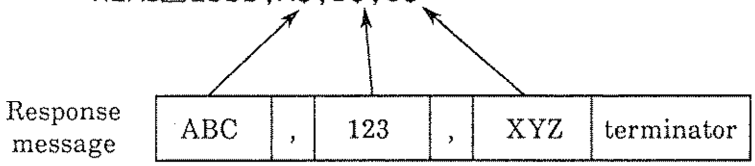
Memory 0	ABC (LF)
Memory 1	123 (LF)
Memory 2	XYZ (LF)

<terminator> is LF.

After executing statements shown on the above left, the contents of the dual-port memory are as shown on the above right.

When these data are read using "PMY?" command, the following contents are stored in variables A$, B$, and C$, respectively.

```
PUT"PMY?␣Ø,3"
READ␣1ØØØ,A$,B$,C$
```



Response message

ABC	,	123	,	XYZ	terminator

- Comma <,> in dual-port memory

The output data for PMY? is assumed to be everything from the beginning to the <termination> code of the specified memory number. The output data includes the memory contents up to (but not including) the terminator. If a comma <,> is included in the contents, it indicates the presence of output data.

In contrast, data in the READ statements for the PTA and controller are separated by commas and sequentially assigned to data variables. Therefore, the number of output variables generated by the PMY? command may be different from the number of variables required for the corresponding statement.

Example:

Contents of dual-port Memory

Memory 0	ABC, DEF (LF)
Memory 1	XYZ (LF)

Execute the statements shown below to read the contents of the dual-port memory at addresses 0 and 1.

```
PUT␣"PMY?␣Ø,2"

READ␣1ØØØ,A$,B$
```

The ABC represents data for variable A$ and the DEF represents data for variable B$. The contents of the memory 0 are separated by a comma (,). This comma separates the data into two data values. Consequently, the XYZ data in the memory 1 is not read. Therefore, the number of input variables in the READ statement must be set to three.

6.6 Transferring, Loading and Saving a PTA Program

PTA application program can be input to the MS8604A from a personal computer using the GPIB1 port. Application programs stored in the MS8604A can also be read to and filed in a personal computer via the GPIB.
Note that the MS8604A must be set to "device" (not controller) when this operations are carried out.
Sample programs are explained using programs for the NEC PC9800 series personal computer N88 - BASIC.

6.6.1 Transferring programs from a personal computer to the PTA

```
100 '-------------------------
110 '-                       -
120 '-   PC9801 → MS8604A    -
130 '-      DOWNLOAD         -
140 '-   SAMPLE PROGRAM      -
150 '-                       -
160 '-------------------------
170 '
180 '
190 PRINT "PC9801 → MS8604A"
200 LET SPA=1
210 ISET IFC
220 ISET REN
230 CMD DELIM=2         .................................. Specifies lf as the terminator.
240 CMD TIMEOUT=5
250 '
260 PRINT @SPA;"PTA 1" ......................................... Turns on the PTA.
270 PRINT @SPA;"PTL 1" ...................................... Turns on PTL mode.
280 PRINT "*** DRIVE NUMBER? ***"
290 INPUT DRV$
300 PRINT "*** FILE NAME? ***"
310 INPUT NAM$
320 FILE$=DRV$+":"+NAM$
330 OPEN FILE$ AS #1 ....................................... Opens the program file.
340 IF EOF (1) THEN 400
350 '
```

```
360 LINE INPUT #1, DAT$
370 PRINT @SPA;DAT$  ................................ Transfers one line of program.
380 GOTO 340
390 '
400 CLOSE            .......................................... Closes the file.
410 PRINT @SPA;"PTL 0"
420 PRINT "COMPLETED !!"
430 IRESET REN
440 END
```

Notes:

- The file storing the PTA program must be created in the personal computer in advance.
- When an error was generated and transfer did not end normally, send "PTL0" (turn off PTL mode) to the MS8604A.
- A PTA program in the execute state cannot be transferred.

Some personal computers used may not operates normally. In this case, put a WAIT statement between the lines 400 and 410.

Example:

```
405 FOR I=1 TO 1000:NEXT I
```

6.6.2 Transferring programs from the PTA to a personal computer

```
100 '-------------------------
110 '-                       -
120 '-   MS8604A → PC9801    -
130 '-        UPLOAD         -
140 '-   SAMPLE PROGRAM      -
150 '-                       -
160 '-------------------------
170 '
180 '
190 PRINT "MS8604A → PC9801"
200 LET SPA=1
210 ISET IFC
220 ISET REN
230 CMD DELIM=2        ................................ Specifies LF as the terminator.
240 CMD TIMEOUT=5
250 '
260 PRINT @SPA;"PTA 1" ........................................ Turns on the PTA.
270 '
280 PRINT "*** DRIVE NUMBER? ***"
290 INPUT DRV$
300 PRINT "*** FILE NAME? ***"
310 INPUT NAM$
320 FILE$=DRV$+":"+NAM$
330 OPEN FILE$ AS #2 .................................... Opens the program file.
340 PRINT @SPA;"PTL 2" .................................... Turns on PTL mode.
350 '
360 LINE INPUT @SPA;DAT$
370 A$=MID$(DAT$,1,3)
380 IF A$="END" THEN 420 ............................ Outputs "END" at the end.
390 PRINT #2,DAT$      ................................ Transfers one line of program.
400 GOTO 360
410 '
```

```
42Ø CLOSE              ........................................... Closes the file.
43Ø '
44Ø PRINT "COMPLETED !!"
45Ø IRESET REN
46Ø END
```

Notes:

- Provide the transfer program in the PTA program area in advance.
- The transferred program is stored in the specified file.
- A PTA program in the execute state cannot be transferred.

(Blank)

SECTION 7
RS-232C IN PTA (OPTION 02)

TABLE OF CONTENTS

(Blank)

7.1 General

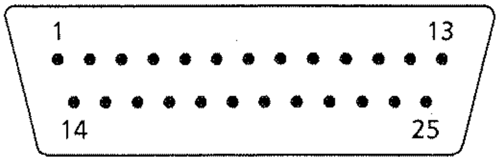
The MS8604A digital mobile transmitter tester has the two-port GPIB and parallel I/O as standard equipment. If the RS-232C interface (option) is used, external devices can be controlled with the RS-232C and data can be sent and received.

7.1.1 RS-232C port function

The RS-232C port can be used as a controller port for an external device connected to the RS-232C port, independently of GPIB1 and GPIB2. When using the RS-232C port, be sure to turn on PTA.

7.2 Interface Specifications

(1) Connector pin layout



Pin numbers (cable connector)

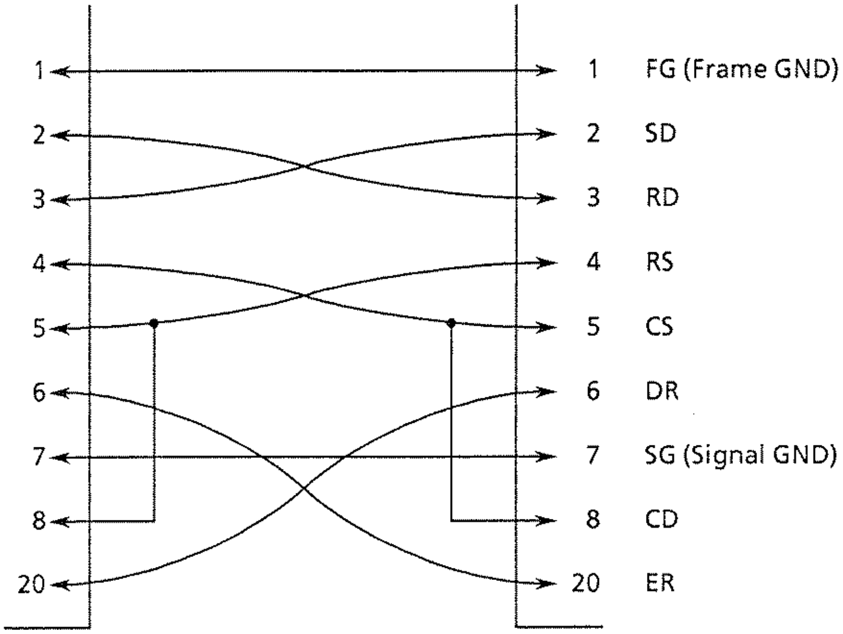
(2) Correspondence between pin numbers and signal cables

Pin No.	Signal name	Input or output	Pin No.	Signal name	Input or output
1	GND	—	16	GND	—
2	SD	Output	17	Unused	—
3	RD	Input	18	GND	—
4	RS	Output	19	Unused	—
5	CS	Input	20	ER	Output
6	DR	Input	21, 22	Unused	—
7	GND	—	23	GND	—
8	CD	Input	24	Unused	—
9 to 15	Unused	—	25	GND	—

(3) Signal cables

- SD : Send data
  Indicates serial send (transmit) data, and sends terminal data to the modem.
- RD : Receive data
  Indicates serial receive data, and sends data received by the modem to the terminal.
- RS : Request for sending
  The terminal requests the modem to send data.
- CS : Ready for sending
  Indicates that the modem can send data.
- DR : Data set ready
  Indicates that the modem can operate.
- CD : Received carrier detection
  Indicates that the modem is receiving a signal (carrier).
- ER : Data terminal ready
  Informs the modem that the terminal can send and receive data.
- GND : Grounding
  Security grounding or cable shield.

(4) Cable wiring



(5) Communication control specifications

Item	Setting
Communication method	Full-duplex
Communication control method	READY/BUSY
Transfer rate (bps)	300, 600, 1200, 2400, 4800, 9600
Data bit	7 bit, 8 bit
Parity	None (OFF), odd number (ODD), even number (EVEN)
Start bit	1 bit
Stop bit	1 bit, 1.5 bit, 2 bit

7.3 Device Messages in PTA

(1) PTA dedicated GPIB commands (Used by WRITE 1000, READ 1000)

When setting or reading parameters with PTA, use the WRITE 1000 or READ 1000 statement and send GPIB format device messages.

The following messages can be sent by the PTA besides the GPIB commands in the MS8604A.

Function	Message		
Port Switching	Control	`PORT␣1`	; Selects GPIB1 as the PTA control port.
		`PORT␣2`	; Selects GPIB2 as the PTA control port.
		`PORT␣3`	; Selects the RS-232C as the PTA controller port.
	Request	`PORT?`	; Requests the PTA control port.
Event Occurrence DELAY (Clock 1)	Control	`EDLY␣t`	; Sets the DELAY time an event interrupt will occur. t: DELAY time: 1 seconds up to 1 hour (in 1 s step)
Event Occurrence TIME (Clock 2)	Control	`ETIM␣t1,t2,t3`	; Sets the time an event interrupt will occur. t3: Seconds: Up to 59 seconds t2: Minutes: Up to 59 minutes t1: Hours: Up to 23 hours
Event Occurrence CYCLE (Clock 3)	Control	`ECYC␣t`	; Sets the cycles an event interrupt will occur. t: Cycle: 1 seconds up to 1 hour (in 0.1 s steps)

Notes:

- For details on the WRITE 1000 and READ 1000 statements, see paragraphs 4.2.20 and 4.2.21.
- For details on event interrupts, see paragraph 4.2.23.
- The control port (for the WRITE, READ, LISTG statements and other GPIB statements supported by the PTA) is the port selected by the PORT command except when these statements are executed with a direct port specification.
  In the initial state, the GPIB1 port is selected as the PTA control port.
- Ports specified by the port switching command are not initialized by PTA → OFF. Port 1 (GPIB1 port) is initialized only when the power is turned on.

(2) GPIB device messages of PTA (for the RS-232C)

Function name	Item name	MS8604A			Remarks
		Program Msg	Query Msg	Response Msg	
Control port select	GPIB1	PORT 1	PORT?	PORT 1	
	GPIB2	PORT 2	PORT?	PORT 2	
	RS-232C	PORT 3	PORT?	PORT 3	
Event occurrence condition	delay	EDLY m	——	——	
	Time	ETIM m1, m2, m3	——	——	
	Cycle	ECYC m	——	——	
RS-232C Interface	Transfer rate (300)	BAUD 300	BAUD?	300	
	(600)	BAUD 600	BAUD?	600	
	(1200)	BAUD 1200	BAUD?	1200	
	(2400)	BAUD 2400	BAUD?	2400	
	(4800)	BAUD 4800	BAUD?	4800	
	(9600)	BAUD 9600	BAUD?	9600	
	Parity bit (OFF)	PRTY OFF	PRTY?	OFF	
	(ODD)	PRTY ODD	PRTY?	ODD	
	(EVEN)	PRTY EVEN	PRTY?	EVEN	
	Data bits (7 bit)	DTAB 7	DTAB?	7	
	(8 bit)	DTAB 8	DTAB?	8	
	Stop bit (1 bit)	STPB 1	STPB?	1	
	(1.5 bit)	STPB 1.5	STPB?	1.5	
	(2 bit)	STPB 2	STPB?	2	
	Time-out time	TOUT t	TOUT?	t	
	Status notification	——	RSST?	n	

7.4 Functions of RS-232C

(1) Program listing

The LISTG command lists programs from the RS-232C port to an external printer.

(2) Command sending

System subroutine CALLGPIB() sends a command to an external device. This command is sent to control the measuring instrument conforming to IEEE 488.2 connected using the RS-232C. This control function corresponds to that of GPIB.

`CALL GPIB(3,3)`: Outputs the "GTL+ <CR+LF>" command.

`CALL GPIB(3,4)`: Outputs the "DCL+ <CR+LF>" command.

`CALL GPIB(3,5)`: Outputs the "LLO+ <CR+LF>" command.

Note: See Paragraph 5.2.19, "Interface control subroutines" for details of CALL GPIB ( ).

(3) Data sending

The WRITE statement sends data to a device connected to the RS-232C.

```
WRITE␣M,Variable[:Format][,Variable[:Format]····]
```

Output data (character constants available)

External device address (numeric constant or variable used)

(4) Data receiving

The READ statement receives data from a device connected to the RS-232C.

```
READ␣M,Variable [,Variable·····]
```

Received data is input in the variable.

External device address (numeric constant or variable used)

(5) Time-out

The time-out time is input as five seconds (initial value).
Use the following GPIB command to change the time-out time:

`TOUT␣t` t=0 to 255 seconds (second unit)

If t=0 is specified, no time-out is set.

(6) Terminating Codes for READ/WRITE Statements

The following terminating codes are used for the RS-232C port.

Send terminators

<Port> command	Terminator code
WRITE LISTG	CR+LF fixed

Receive terminators

<Port> command	Terminator code
READ	LF or CR+LF

(7) RS-232C status notification

Query command "RSST?" can be used to read a status of the RS-232C port as a decimal character string from 0 to 255.

Bit	Status	Explanation
7	CD line	0: OFF, 1: ON
6	A framing error occurred.	The bit is cleared after reading.
5	An overrun error occurred.	The bit is cleared after reading.
4	A parity error occurred.	The bit is cleared after reading.
3	A receiving buffer overflowed.	The bit is cleared after reading.
2	Sending available/unavailable	0: Available, 1: Unavailable
1	Receiving available/unavailable	0: Available, 1: Unavailable
0	Receiving data exists/does not exist	0: Does not exist, 1: Exists

(Blank)

SECTION 8
PTA ERROR MESSAGES

TABLE OF CONTENTS

(Blank)

An error message is displayed when an error is detected in the PTA command or program.

There are two types of errors;an execution-stop error (fatal: F) and an execution-continuable error (warning: W).

- Execution-stop error (F:Fatal) :
  This type of error stops the execution of the program unconditionally.
- Execution-continuable error (W:Warning error) :
  When there is no ERROR statement in the line next to the line where this type of error occurs, the execution stops; but if there is an ERROR statement, execution continues.
  And also, error interruption process can continue the execution.

8.1 Error Message Format

The error message is displayed in the following format.

```
ERROR Error level Error number[,Error-occurrence line number]
```

This is displayed at the program execution.

8.2 ERROR Statement

(1) Function

For an execution-continuable error generated at program execution, execution can be continued by using the ERROR statement.

An ERROR statement can be programed over several lines.

(2) Format

```
ERROR (210,1000)
```

Next program line to be executed

Error number

This statement means that when the error (generated in the previous line) corresponds to the error number 210,the program of line 1000 is executed.
When it does not correspond, the error message is displayed and execution stops.

(3) Example

```
10 X=0
20 Y=100/X
30 ERROR(210,100)    ; If the error (210: the divisor is 0) occurs, jump to line 100.
40 Y=Y+ 50
       ⋮
```

8.3 ERRMAIN Statement

(1) Function

To branch to the main routine whenever a execution continuable ERROR occurred, use the ERRMAIN statement.

(2) Format

```
ERRMAIN (error number)
```

(3) Example

```
  1Ø INPUT A
  2Ø GOSUB 1ØØØ
  3Ø  :
      :
1ØØØ WRITE 217, A
1Ø1Ø ERRMAIN (222)   ; If the error (222) occurs because the data of WRITE statement can not
                       output, the program returns to the main routine.
      :
```

Note: If the ERRMAIN statement has been executed in the highest level of the routine, ERROR 213 is generated.

8.4 Error Processing Subroutines

8.4.1 ON ERROR statement

(1) Function

Registers the subroutine to branch (interrupt) to when an error occurs.

(2) Format

```
ON ERROR line number (or *label)
```

After executing this statement and an error that is possible to continue execution occurs, an interrupt occurs and the error processing subroutine is executed from the line number (or label) specified.

8.4.2 OFF ERROR statement

(1) Function

Releases the registered subroutine to branch (interrupt) to when an error occurs.

(2) Format

```
OFF ERROR
```

After executing this statement, error interrupts will not occur.

8.4.3 Returning from error processing subroutines (RETERR, RETRY, RESUME and GIVE UP statements)

(1) Function

Returns from an error interrupt.

(2) Format

RETERR	(Continues from the statement following the statement where the error occurred.)
RETRY	(Continues reexecuting from the statement caused the error.)
RESUME	(Continues from specified line.)
GIVEUP	(Stops program execution.)

Note: See paragraphs 4.2.38 to 4.2.41.

8.4.4 ERRREAD (m) function

(1) Function

Reads the line where the error occurred and the error code in the middle of an error processing subroutine.

(2) Format

```
V=ERRREAD (Ø)     (Error code)
V=ERRREAD (1)     (Line where error occurred)
```

(3) Example

```
100 ON ERROR 200          ; Jumps to line 200 on error
110 INPUT X
120 Y=100/X
130 PRINT Y
140 GOTO 110
150 STOP
200 C=ERRREAD(Ø)
210 IF C=210 GOSUB 300    ; For "Divide by zero", continues to execute from line 130
                            displaying "ERROR /0".
220 IF C<>210 GIVEUP      ; On other errors, stops program execution
230 RETERR
300 PRINT "ERROR/Ø"
310 RETURN
```

8.5 Error List

Table 8-1 shows the error number and error cause. In the table, F (Fatal) denotes the execution-stop error and W (Warning) denotes the execution-continuable error.

Table 8-1 PTA Error List

Error No.	Cause of error	W,* F**
0	[↵] key pressed but no commands or statement input	F
1	Number of characters (representing variable) exceeds 8, or number of characters (representing program name) exceeds 6.	W
2	Format of numeric constant in correct Example : `0..1 4.5 EE2`	W
3	Too many input digits, or value of numeric constant too large or too small (Format of numeric constant incorrect)	W
4	Format of character string constant incorrect Example : `A$="ABC`	W
5	Format incorrect Example : `PRINT A:G6.2`	W
6	Statement cannot be interpreted (command format error) Example : `GOTO ABC`	W
7	Statement insufficiently described Example : `GOTO`	W
8	Statement excessively described Example : `GOTO 1ØØ,2ØØ`	W
9	Number of variables exceeds 256 (Up to 256 user-defined variables can be written)	W
10	Character cannot be interpreted Example : `-1ØØ`	W
11	Format (of binary or hexadecimal constant) incorrect Example : `8#=# 11Ø`	W
12	Value (of binary or hexadecimal constant) too large Binary constant : up to 8 characters Hexadecimal constant : up to 2 characters Example : `8#=#1ØØØØØØØØ`	W

*W : Execution-continuable error (Warning)

* *F : Execution-stop error (Fatal error)

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
13	Number of format digits too large Example : `PRINT A:F6.5`	W
14	Command operand cannot be interpreted Example : `LIST A,B`	W
15	Command operand insufficient Example : `LISTG`	W
16	Command operand excessive Example : `DELETE 1Ø,1ØØ,3ØØ`	W
17	Line number exceeds 65535 (Program line number is 1 to 65535)	W
20	Program on a line too long to assemble	W
21	Undefined-line-number label used as command operand	W

Note: Errors 0 to 21 may occur during program input or command execution. Errors 6 to 8, however , may also occur during statement execution.

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
101	Value of command operands 1 and 2 incorrect Example : LIST 1ØØ,1Ø	F
102	Program exceeds memory capacity	F
103	No Line number or program, designated by command (LIST, LISTG, DELETE, RENUM, and SAVE commands)	F
104	Since number of GOTO or GOSUB statements excessive (>100), RENUM statement cannot be executed	F
105	Since line number (specified by GOTO or GOSUB operand) not found, RENUM statement cannot be executed	F
111	Line number exceeds 65535 when RENUM and PCOPY statements executed	F

Note: Errors 101 to 105 and 111 may occur during command execution.

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
120	Media write-protected	W
121	Media not installed	W
122	Media memory overflow	W
123	Specified program not stored in media	W
124	Media faulty	W
125	Memory type incorrect	W
126	Media formatting incorrect	W
127	Media not formatted	W
150	Label is not defined or defined more than once	F
151	No DATA statement	F
180	Error of the command transmitted from PTA to main frame	W

Note: Errors 120 to 127 may occur when a command or statement attempts to access the media (PMC or FD).

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
201	Program cannot be resumed (CONT command)	F
202	Specified line number missing RUN command executed without program (RUN, CONT commands and GOTO, GOSUB statements)	W
203	Array subscript (in DIM statement) incorrect (The array subscript must be from 1 to 1024; the bit array subscript must be from 1 to 8, and the character array subscript must be from 1 to 255.)	W
204	Used as simple, or system variables before array declaration by DIM statement	W
205	Array declaration overlapped	W
206	Insufficient variable memory capacity due to program memory overflow	F
207	Arithmetic operation of character data or bit data	W
208	Data-type combination incorrect for conversion	W
209	Overflow or underflow occurred	W
210	Divide by 0	W
211	Value of arithmetic function parameter too large or too small	W
212	Nesting (by subroutine, FOR and NEXT statement) exceeded 10 levels	F
213	No return destination specified for RETURN statement	F
214	Comparison cannot be made by IF statement Right and left side data-type combination incorrect	W

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
215	SOS statement is executed	F
216	No corresponding FOR statement. That is, there are excess NEXT statements. (RUN, CONT command and GOTO, GOSUB statements)	W
217	Input data format (in INPUT statement) incorrect	W
218	Input data (in INPUT statement) insufficient	W
219	Excess amount or too large input data in INPUT statement	W
220	Minus sign used in exponentiation Example : `-1!5`	W
221	Data can not be input in GPIB (Talker device not connected)	W
222	Data cannot be output in GPIB	W
223	Parameter (in the statement) outside range or variable type incorrect Example : `WAIT A$`	W
224	Simple variable includes array subscript	W
225	Array variable has no subscript	W
226	Array-variable subscript out of boundary Note that the subscript range declared in DIM J(5) is J(0) to (4).	W
227	GPIB execution is impossible because the PTA is set as the device	W
228	GPIB execution is impossible because the PTA is set as the controller	W

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
229	STOP statement (to terminate program execution) not specified	W
230	Attempt made to refer to non-referable system variable	W
231	Attempt made to assign non-assignable system variable	W
232	Array variable subscript not numeric	F
233	Parameter (in boolean function) not bit type	W
234	Parameter of FOR statement is character or bit type	W
235	The I/O type specification in the EVENT statement is out of range (0 to 99).	W
236	Variable of NEXT statement does not correspond to that of FOR statement specified before NEXT statement Example : `30 FOR C=···` `90 NEXT D`	W
237	Six or more character constants and variables used in INPUT, PRINT, READ or WRITE statement Example : `PRINT"FREQ",F(C),"Hz","LEVEL",LEV,` `"dBm"`	W
238	Variable type and format type of PRINT or WRITE statement do not agree	W
239	Operand (in LISTG, WRITE or READ statement) outside range (0 to 31) Example : `LISTG 35`	W
240	Variable or constant values of CALL statement or system function outside range	W
241	Variable or constant type of CALL statement or system function incorrect	W
242	System variable used in CALL statement or system function	W

Table 8-1 PTA Error List (Continued)

Error No.	Cause of error	W, F
243	The RETURN or RETMAIN statement was used to return from event or error interrupt processing.	F
244	Media data file not open	W
245	Media data file opened	W
246	Media data already read	W
247	Media data type and variable type combination incorrect (unconvertible)	W
248	Excess amount or too large input data value in READ statement.	W
249	Insufficient input data in READ statement	W
250	Input data format (in READ statement) incorrect	W
251	The RETINT statement was used for something other than event interrupt processing. Or, the GOSUB statement was executed in the middle of event interrupt processing and the RETURN statement to return was not executed, but the RETINT statement was instead.	F
252	The RETERR, RETRY, RESUME, GIVEUP statements were used for something other than error interrupt processing. Or, the GOSUB statement was executed in the middle of error interrupt processing. Or, the RETURN statement to return was not used and one of the above statements was executed.	F
253	The ERRREAD function was executed for something other than error interrupt processing.	F
254	The STATUS function was executed for something other than event interrupt processing.	F

SECTION 9
I/O PORT CONTROL

TABLE OF CONTENTS

(Blank)

9.1 Outline

The I/O port of the MS8604A can be controlled by the PTL (Personal Test Language). Therefore, autohandling and trimming equipment can be easily connected to the I/O port for configuration of automatic measurement and inspection system.

All signals input and output through the I/O port use negative logic.

9.2 Explanation of I/O Port Signals

The names and specifications of the I/O-port connector pins are shown in Table 9-1.

■ GND (Pin 1)

Ground line

■ INPUT 1 (Pin 2)

This is the pulse input pin. The input pulse (negative logic) sets two internal flip-flop (F/F) circuits of OUTPUT 1 and 2.

This signal is mainly used to start measurement by the external controller. The PTA program must be written so that the processing flow changes to the measurement routine when internal F/F is set. The system variable EX0 is used to check and reset the internal F/F.

■ OUTPUT 1 (Pin 3)

This is a latch output pin. This is the output pin of the internal F/F to be set by the pulse input to INPUT 1 or PTA program.

This signal is mainly used as the status signal indicating that measurement or data processing is in progress. The program must be written so that the internal F/F is reset upon termination of the measurement or data processing.

The system variable EX0 is used to set and reset the internal F/F.

■ OUTPUT 2 (Pin 4)

This is a latch output pin. This is the output pin of the internal F/F to be set by the pulse input to INPUT 1 or PTA program.

This signal is mainly used as the status signal indicating that measurement or data processing is in progress. The PTA program must be written so that the internal F/F is reset upon termination of the measurement or data processing.

The system variable EX0 is used to set and reset the internal F/F.

■ Output Port A: A0 to A7 (Pins 5 to 12)

These are latch output pins. 8-bit data can be output by a program. (No synchronizing or strobe signals can be output.)

The system variable IOA is used to output data to this Output Port A.

■ Output Port B: B0 to B7 (Pins 13 to 20)

These are latch output pins. 8-bit data can be output by a program. (synchronizing or strobe signals cannot be output.)

The system variable IOB is used to output data to this Output Port B.

■ I/O Port C: C0 to C3 (Pins 21 to 24)

These are status-input/latch-output pins which form a 4-bit I/O port. When this port is set to the output mode, the write-strobe signal (Pin 31) is output.

The system variable IOC is used to input/output the data.

The system variable EIO is used to determine whether this port is set to the input mode or output mode.

■ I/O Port D: D0 to D3 (Pins 25 to 28)

These are the status-input/latch-output pins which form a 4-bit I/O port. When this port is set to the output mode, the write-strobe signal (Pin 31) is output.

The system variable IOD is used to input/output the data.

The system variable EIO is used to determine whether this port is set to the input mode or output mode.

■ I/O Port C status (Pin 29)

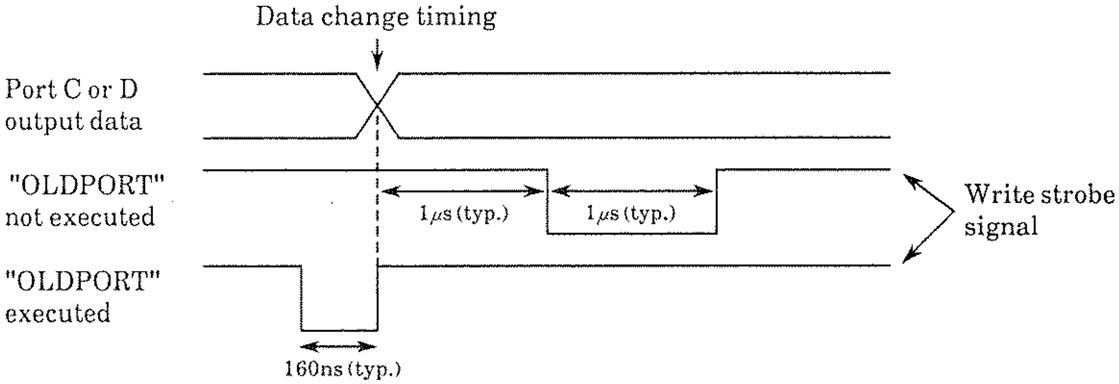
This is a status line which becomes low level when the I/O Port C is set to the input mode by the system variable EIO and becomes high level when the I/O Port C is set to the output mode.

■ I/O Port D status (Pin 30)

This is a status line which becomes low level when the I/O Port D is set to the input mode by the system variable EIO and becomes high level when the I/O Port D is set to the output mode.

■ Write strobe signal (Pin 31)

This is a pulse output pin. A pulse is output when data is output from either I/O port C or D.
The write strobe signal pulse generation timing can be switched by executing an "OLDPORT" statement.



■ Interruption signal (32 pin)

Interruption signal input terminal. When the pulse signal is input to this terminal, a hardware interruption is issued to the PTA.

Table 9-1 I/O Port Specifications

Pin. No.	Name	Specifications	System variable name
1	GND	Ground	—
2	INPUT 1	TTL level, negative logic, pulse input, pulse width ≧ 1 μs	EX0
3	OUTPUT 1	TTL level, negative logic, latch output	
4	OUTPUT 2		
5	Output port A 0	TTL level, negative logic, latch output	IOA
6	Output port A 1		
7	Output port A 2		
8	Output port A 3		
9	Output port A 4		
10	Output port A 5		
11	Output port A 6		
12	Output port A 7		
13	Output port B 0	TTL level, negative logic, latch output	IOB
14	Output port B 1		
15	Output port B 2		
16	Output port B 3		
17	Output port B 4		
18	Output port B 5		
19	Output port B 6		
20	Output port B 7		
21	I/O port C 0	TTL level, negative logic, state input/latch output	IOC
22	I/O port C 1		
23	I/O port C 2		
24	I/O port C 3		
25	I/O port D 0	TTL level, negative logic, state input/latch output	IOD
26	I/O port D 1		
27	I/O port D 2		
28	I/O port D 3		
29	I/O port C status	TTL level, input mode: LOW, output mode: HIGH	EIO
30	I/O port D status		
31	Write-strobe signal	TTL level, negative logic, pulse output	(Note 1)
32	Interrupt signal	TTL level, negative logic	—
33	(NC) (Note 2)		—
34	+5V output	Max. 100 mA	—
35	(NC) (Note 2)		—
36	(NC) (Note 2)		—

Note 1 : Pulses are generated when a data is output from I/O ports C and D.
Note 2 : NC means no connection.

9.3 System Variables for Accessing I/O Ports

The following six system variables are used to access the I/O ports:

EX0
IOA
IOB
IOC
IOD
EIO

These system variables are described below. Setting in the tables means data assignment to the system variable and Read means reading of a data from the system variable.

- EX0 .... This system variable is used to set the states of OUTPUT1 and OUTPUT2 and read the INPUT1 state.

  The variable type is numeric. The meanings of the data for setting/read to EX0 are shown in Table 9-2.

Table 9-2 Setting and read to EX0

Data	Read	Setting
0	INPUT 1 reset	OUTPUT 1 reset (= " H ") OUTPUT 2 reset (= " H ")
1	INPUT 1 set	OUTPUT 1 set (= " L ") OUTPUT 2 reset (= " H ")
2		OUTPUT 1 reset (= " H ") OUTPUT 2 set (= " L ")
3		OUTPUT 1 set (= " L ") OUTPUT 2 set (= " L ")

  While the power is turned on, if the [RESET] key is pressed when the PTA is on, OUTPUT1, OUTPUT2, and INPUT1 are reset.

  INPUT1 and OUTPUT1 are linked. When a signal is input to the INPUT1 terminal from the outside, OUTPUT1 state is set. Resetting of OUTPUT1 switches INPUT1 to the reset state.

- IOA .... This system variable is used to output 8-bit data to Output Port A.
  The IOA variable type is bit.

- IOB .... This system variable is used to output 8-bit data to Output Port B.
  The IOB variable type is bit.

- IOC .... This system variable is used to input/output 4-bit data to and from I/O Port C.

  The IOC variable type is bit.

  The system variable EIO is used to switch the input/output mode.

- IOD .... This system variable is used to input/output 4-bit data to and from I/O Port D.

  The IOD variable type is bit.

  The system variable EIO is used to switch the input/output mode.

While the power is turned on, if the [RESET] key is pressed when the PTA is on, the IOA to IOD output registers are initialized to "H" level.

- EIO .... This system variable is used to set I/O Ports C and D either to the input mode or output mode.

  The EIO variable type is numeric.

  The meanings of the data for setting and read to EIO are shown in Table 9-3.

Table 9-3 Setting and read to EIO

Data	Setting/read
0	Port C : Input mode Port D : Input mode
1	Port C : Output mode Port D : Input mode
2	Port C : Input mode Port D : Output mode
3	Port C : Output mode Port D : Output mode

- While the power is turned on, if the [RESET] key is pressed when the PTA is on, ports C and D are input mode.

9.4 Interruption by I/O Ports

The PTA of the MS8604A can be interrupted by driving pin 32 to TTL low level. When an interruption is generated at permitted state of I/O port interruption by ENABLE EVENT statement, program execution branches to the line number specified by ON EVENT statement.

The IOEN and ON IO GOTO (GOSUB) statements can also perform interrupt processing.

See paragraph 4.2.23 to 4.2.26 for details on using I/O interruption.

Example :

```
  1Ø ON EVENT 41,1ØØØ
  2Ø ENABLE EVENT 41,Ø,Ø,Ø,1
  3Ø        (              }
            )              } Measurement routine
       GOTO 3Ø             }
1ØØØ REM"PRINT OUT"
1Ø1Ø PRINT LMAX, "dBm"
1Ø2Ø RETINT
```

Before an interruption is generated, the above program repeatedly executes the measurement routine. When interruption is generated, program execution jumps to line 1000.

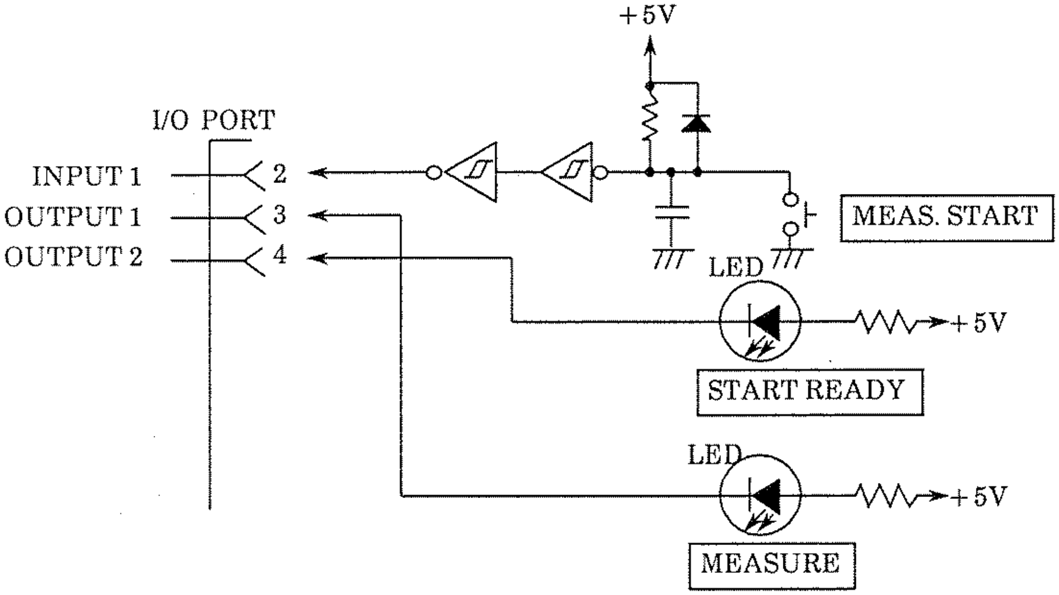
When an interruption completed, control is returned to the previous measurement routine.

9.5 Application Examples

(1) Using INPUT 1, OUTPUT 1, and OUTPUT 2

An example is shown below where INPUT 1 is used as an input for measurement start (MEAS.START) ; OUTPUT 1 is used to indicate that measurement and data processing are in progress (MEASURE), and OUTPUT 2 is used as an indicator for measurement-start wait (START READY).

Circuit example 1:



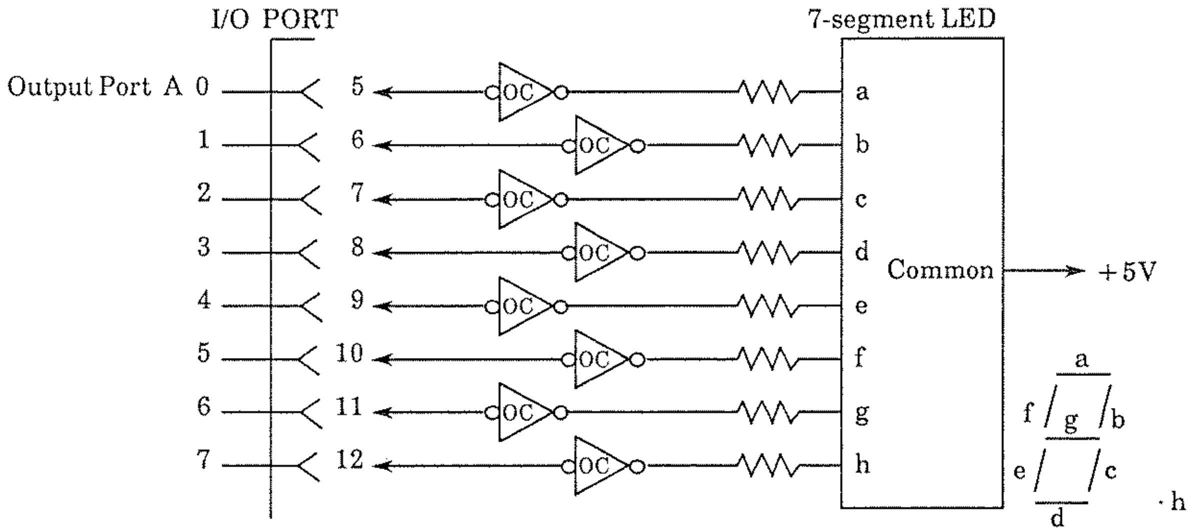
Program example 1:

```
  1Ø EXØ=Ø
   ⋮  ⌇
   ⋮  Measurement-preparation routine
   ⋮  ⌇
 1ØØ EXØ=2 ........................ START READY lamp ON
 11Ø C=EXØ ........................ Wait for MEAS. START switch to be pressed.
 12Ø IF C=Ø GOTO 11Ø ............. Wait for MEAS. START switch to be pressed.
 13Ø EXØ=1 ........................ START READY lamp OFF, MEASURE lamp ON
   ⋮  ⌇
   ⋮  Measurement and data processing routines
   ⋮  ⌇
 2ØØ EXØ=Ø ........................ MEASURE lamp OFF
 21Ø STOP
```

(2) Using Output Port A or Output Port B

An example where a 7-segment numeric display LED is connected to Output Port A for displaying numerics, is shown below.

Circuit example 2:



OC: Open-collector output

Program example 2:

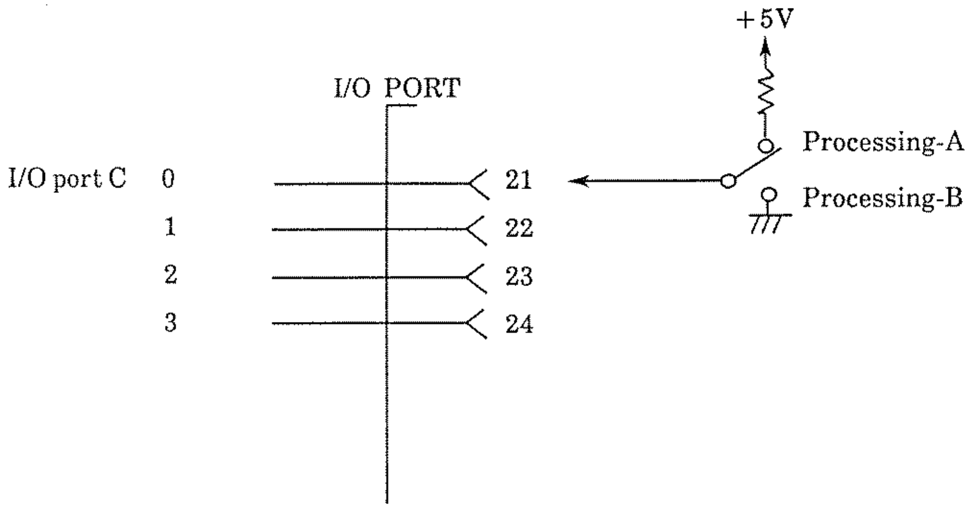
An example program where 0 to 9 are sequentially displayed each time the MEAS. START switch shown in Circuit example 1 is pressed, is shown below.

```
 1Ø DIM D(1Ø)
 2Ø D(Ø)=$5C
 3Ø D(1)=$Ø6
 4Ø D(2)=$5B
 5Ø D(3)=$4F
 6Ø D(4)=$66      .............   Definition of LED segment data
 7Ø D(5)=$6D
 8Ø D(6)=$7D
 9Ø D(7)=$27
1ØØ D(8)=$7F
11Ø D(9)=$6F
12Ø IOA=$Ø ...................   LED OFF
13Ø N=Ø
14Ø EXØ=Ø
15Ø C=EXØ            .......   Checking MEAS.START switch
16Ø IF C=Ø GOTO 15Ø
17Ø IOA=D(N) .................   Display output
18Ø N=N+1
19Ø IF N=<1Ø GOTO 14Ø
2ØØ STOP
```

(3) Using I/O Port C or I/O Port D

An example where the processing routine is changed depending on whether bit 0 of I/O Port C is 0 or 1, is shown below.

Circuit example 3:



Program example 3:

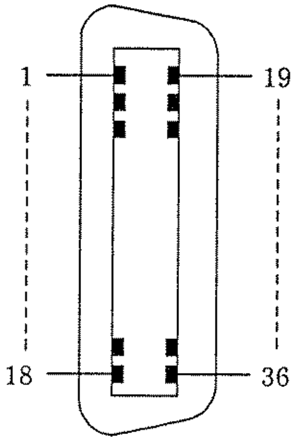
An example is shown below where a switch is set to processing A or processing B. By pressing the MEAS. START switch shown in Circuit Example 1, Port C is checked and the processing is branched.

```
 1Ø EIO=Ø
 2Ø EXØ=Ø
 3Ø C=EXØ
 4Ø IF C=Ø GOTO 3Ø
 5Ø D=IOC
 6Ø IF D=1 GOTO 2ØØ
      ⋮  ⌇
      ⋮ Processing A
      ⋮  ⌇
1ØØ STOP
2ØØ REM
      ⋮  ⌇
      ⋮ Processing B
      ⋮  ⌇
25Ø STOP
```

9.6 Connector Name and Appropriate Connector

The name of the I/O port connector with the MS8604A is RC30-36R (Hirose Electric, Japan), and, appropriate connector for the RC30-36R is RC30-36P (Hirose Electric, Japan).

Connector pin No. assignment



SECTION 10
PTL INDEX

(Blank)