

# **F300 Visual Inspection System OMRON Vision Language (OVL)**

## **Reference Manual**

*Revised March 1994*



## **Notice:**

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify precautions in this manual. Always heed the information provided with them. Failure to heed precautions can result in injury to people or damage to the product.

**DANGER!** Indicates information that, if not heeded, is likely to result in loss of life or serious injury.

**WARNING** Indicates information that, if not heeded, could possibly result in loss of life or serious injury.

**Caution** Indicates information that, if not heeded, could result in relative serious or minor injury, damage to the product, or faulty operation.

## **OMRON Product References**

All OMRON products are capitalized in this manual. The word "Unit" is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation "PLC" means Programmable Controller (Programmable Logic Controller) and is not used as an abbreviation for anything else.

## **Visual Aids**

The following headings appear in the left column of the manual to help you locate different types of information.

**Note** Indicates information of particular interest for efficient and convenient operation of the product.

**1, 2, 3...** 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

## **© OMRON, 1992**

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.



# TABLE OF CONTENTS

## PART I, Version 1.00

### SECTION 1

<b>List of Commands and Functions</b> .....	<b>1</b>
1-1 OVL Alphabetical List .....	2
1-2 OVL Instruction List .....	11

### SECTION 2

<b>OVL Syntax</b> .....	<b>19</b>
2-1 Line Format .....	20
2-2 OVL Characters and Symbols .....	20
2-3 Constants .....	21
2-4 Variables .....	23
2-5 Type Conversion .....	25
2-6 Operations and Expressions .....	27
2-7 Interrupts .....	32
2-8 Labels .....	32

### SECTION 3

<b>Screen Operations</b> .....	<b>35</b>
3-1 Flow of Image Data .....	36
3-2 Display and Measurement Block Diagrams .....	40
3-3 Flow of Measured Data .....	41
3-4 Plane and Frame Memory .....	43
3-5 Windows .....	44
3-6 Drawing Density and Drawing Modes .....	46
3-7 Run Length .....	47
3-8 Labelling .....	48
3-9 Scrolling .....	49
3-10 Fill Measurement .....	50
3-11 Image Cut-off .....	50
3-12 Scan Measurement .....	51

### SECTION 4

<b>Reference</b> .....	<b>53</b>
------------------------	-----------

# **PART II, Version 2.00**

## **SECTION 1**

### **OVL Version 2.00 Improvements . . . . . 169**

1-1 Overview of Improvements . . . . . 170

1-2 Additional OVL Capabilities . . . . . 172

## **SECTION 2**

### **Reference . . . . . 183**

## **SECTION 3**

### **Sample Programs . . . . . 203**

3-1 Determination of Multiple Windows' ON/OFF Status . . . . . 204

3-2 Shape Inspection using Window Enlargement/Reduction . . . . . 208

### **Appendices . . . . . 213**

A. Table of Error Messages . . . . . 213

B. Reserved Words . . . . . 215

C. Induction Functions . . . . . 217

### **Index . . . . . 219**

### **Revision History . . . . . 229**

## ***About this Manual:***

This manual describes the OMRON Vision Language (OVL) used with the F300 Visual Inspection System and includes the sections described below.

Please read this manual completely and be sure you understand the information provided before attempting to operate the F300 Visual Inspection System and use the OVL.

### ***PART I***

#### ***Version 1.00***

**Section 1** provides a listing of commands and functions. There is a list ordered alphabetically and a list ordered by instructions.

**Section 2** provides the basic OVL syntax required before programming.

**Section 3** provides general information on screen manipulation.

**Section 4** provides detailed information on the commands and functions. Examples are also provided.

### ***PART II***

#### ***Version 2.00***

**Section 1** describes the additional functions and improvements found in OVL Version 2.00.

**Section 2** provides detailed information on the new Version 2.00 commands and functions. Examples are also provided.

**Section 3** provides sample programs using the new Version 2.00 commands and functions.

This manual also contains three appendices. **Appendix A** contains a listing of error messages, **Appendix B** contains a listing of reserved words, and **Appendix C** contains a listing of induction functions.

<p><b>WARNING</b> Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each section in its entirety and be sure you understand the information provided in the section and related sections before attempting any of the procedures or operations given.</p>
---

# **PART I**

## **Version 1.00**

### **SECTION 1**

#### **List of Commands and Functions**

This section provides a listing of commands and functions. There is a list ordered alphabetically and a list ordered by usage.

1-1	OVL Alphabetical List .....	2
1-2	OVL Instruction List .....	11



## 1-1 OVL Alphabetical List

Command/Function	Operation	Instruction	Type	Page	
A	ABS	Determines an absolute value.	Calculation	Function	54
	AKCNV\$	Converts 1-byte characters to 2-byte characters.	Japanese character string operation	Function	54
	ARC	Draws an arc.	Graphic control	Command	54
	ASC	Determines the character code.	1-byte character string operation	Function	55
	ATN	Determines the arctangent (arctan) of a value.	Calculation	Function	55
	ATTR\$	Determines the write protect attribute of a file.	File control	Function	55
	AUTO	Automatically generates line numbers.	Program editing	Command	56
	AUTOLVL	Determines the binary level.	Image processing	Function	56
B	BACKDISP	Sets the image display outside the window.	Display control	Command	56
	BCOPY	Copies a binary image.	Special graphic display control	Command	57
	BCOPY2	Makes enlarged and reduced copies of binary images.	Special graphic display control	Command	58
	BEEP	Turns the buzzer on and off.	Special	Command	58
	BOX	Draws a rectangle.	Graphic control	Command	59
	BUSY	Controls BUSY signal output status.	I/O port I/O control	Command	59
C	CALL	Calls a structural subroutine.	General	Command	60
	CAMCHK	Determines the camera connection status.	Camera/strobe control	Function	60
	CAMERA	Switches cameras.	Camera/strobe control	Command	60
	CAMMODE	Selects the image output from a Camera I/F Unit.	Camera/strobe control	Command	60
	CDBL	Converts to a double-precision value.	Calculation	Function	61
	CHAIN	Transfers control to a specified program.	General	Command	61
	CHANGE	Changes the measuring conditions.	Scene	Command	62
	CHDIR	Changes the current directory.	File operation	Command	62
	CHR\$	Converts a character code to a character.	1-byte character string operation	Function	63
	CINT	Converts to an integer.	Calculation	Function	63
	CIRCLE	Draws a circle.	Graphic control	Command	64
	CLEAR	Initializes variables.	Memory management	Command	65
	CLNG	Converts to a long integer.	Calculation	Function	65
	CLOSE	Closes an open file.	File control	Command	65
	CLS	Clears characters and graphics.	Graphic control	Command	66
	COLOR	Changes the character attributes.	Text display control	Command	66
	COLOR@	Changes the character attributes within a specified region.	Text display control	Command	66
	COM ON/OFF/STOP	Controls interrupts from the RS-232C.	RS-232C communication	Command	67
COMMON	Transfers variables.	General	Command	67	
CONSOLE	Sets the text display mode.	Text display control	Command	68	

Command/Function	Operation	Instruction	Type	Page	
C	CONT	Continues execution of a program.	Program execution control	Command	68
	COS	Determines the cosine of an angle.	Calculation	Function	69
	CSNG	Converts to a single-precision value.	Calculation	Function	69
	CSRLIN	Determines the cursor line position.	Text display control	Function	69
	CURSOR	Draws a cross cursor.	Graphic control	Command	69
	CVD	Converts an 8-byte character string to a double-precision real number.	Calculation	Function	70
	CVI	Converts a 2-byte character string to an integer.	Calculation	Function	71
	CVL	Converts a 4-byte character string to a long integer.	Calculation	Function	71
CVS	Converts a 4-byte character string to a single-precision real number.	Calculation	Function	71	
D	DATA	Defines the data read with the READ command.	General	Command	72
	DATE\$	Displays and sets the date.	Time/date control	Function, Command	73
	DEF FN	Defines a user function.	General	Command	73
	DEF FN END DEF	Defines a user function block.	General	Command	74
	DEFDBL	Declares variables as double-precision real number variables.	General	Command	74
	DEFINT	Declares variables as integers.	General	Command	74
	DEFLNG	Declares variables as long integers.	General	Command	75
	DEFSNG	Declares variables as single-precision real number variables.	General	Command	75
	DEFSTR	Declares variables as character variables.	General	Command	75
	DELETE	Deletes part of a program.	Program editing	Command	76
	DEVICE	Specifies the input and output devices.	Special	Command	76
	DIM	Defines an array variable.	General	Command	76
	DIN	Reads data from an input port.	I/O port I/O control	Function	77
	DISPLAY	Sets the display image.	Display control	Command	77
	DO. . . LOOP REPEAT	Repeats a loop specified number of times.	General	Command	78
	DO. . . LOOP UNTIL	Repeats a loop until a condition is fulfilled.	General	Command	78
	DO. . . LOOP WHILE	Repeats a loop while a condition is fulfilled.	General	Command	79
	DO REPEAT LOOP	Repeats a loop specified number of times.	General	Command	79
	DO UNTIL LOOP	Repeats a loop until a condition is fulfilled.	General	Command	80
	DO WHILE LOOP	Repeats a loop while a condition is fulfilled.	General	Command	80
DOUT	Outputs data to an output port.	I/O port I/O control	Command	80	
DSA	Reads the status of the DSA signal.	I/O port I/O control	Function	81	
DSKF	Determines the free space in the memory card.	File control	Function	81	

Command/Function	Operation	Instruction	Type	Page	
E	EDIT	Selects the mode to edit lines of the program.	Program editing	Command	81
	ELLIPSE	Draws an ellipse.	Graphic control	Command	82
	END	Stops program execution.	General	Command	82
	ENHANCE	Creates LUT data for contrast modification.	Image processing	Command	83
	EOF	Determines the end of the file.	File control	Function	83
	ERASE	Deletes an array variable.	General	Command	83
	ERL	Determines the line number where an error occurred.	Error control	Function	84
	ERR	Determines the error code.	Error control	Function	84
	ERRMSG	Defines the operation when an error occurs.	Error control	Command	84
	ERROR	Generates a pseudo-error.	Error control	Command	84
	ERROUT	Controls the error signal output.	I/O port I/O control	Command	85
	EVENTIN	Sets the image input synchronized with events.	Camera/strobe control	Command	85
	EXIT DEF/ DO/FOR/SUB	Exits a control block.	General	General	86
	EXP	Determines the natural number e raised to an exponential power.	Calculation	Function	87
F	FIELD#	Allocates the variable areas.	File control	Command	87
	FILES	Displays file names.	File operation	Command	88
	FILTDATA	Specifies the line filter factors.	Image processing	Command	88
	FILTER	Sets the image filtering function.	Image processing	Command	88
	FILTERIN	Selects image input to the filter.	Image processing	Command	89
	FIND	Searches for a character string in a program.	Program editing	Command	89
	FIX	Rounds down a value to an integer.	Calculation	Function	90
	FLASH	Controls the strobe and shutter function.	Camera/strobe control	Command	90
	FOR. . . TO. . . STEP ~ NEXT	Repeatedly executes instructions.	General	Command	91
	FRE	Determines the free memory size.	Memory management	Function	91
G	GATE	Controls the GATE signal.	I/O port I/O control	Command	91
	GCOPY	Copies the raw image in VRAM.	Special graphic display control	Command	92
	GCOPY2	Makes enlarged and reduced copies of the raw image.	Special graphic display control	Command	92
	GET#	Reads data from a random file.	File control	Command	93
	GET@	Reads image data to VRAM.	Graphic control	Command	93
	GETBLUT	Reads the binary LUT data.	Image processing	Command	94
	GETDLUT	Reads the display LUT data.	Display control	Command	94
	GETDLVL	Determines the display level of the image.	Display control	Function	95
	GETLUT	Reads the filter LUT data.	Image processing	Command	95
	GOSUB	Branches to a specified subroutine.	General	Command	95
	GOTO/GO TO	Unconditionally jumps to a specified line.	General	Command	96

Command/Function	Operation	Instruction	Type	Page	
H	HELP	Displays help messages.	Program editing	Command	96
	HELP ON/OFF/STOP	Disables, enables, or stops interrupts from the HELP Key.	Key control	Command	96
	HEX\$	Converts a numeric expression to hexadecimal.	1-byte character string operation	Function	97
	HISTGRAM	Reads a density histogram.	Image processing	Command	97
I	IF. . . GOTO ~ ELSE	Evaluates a condition.	General	Command	98
	IF. . . THEN ~ ELSE	Evaluates a condition.	General	Command	98
	IF. . . THEN ~ ELSEIF ~ ELSE ~ END IF	Evaluates a condition.	General	Command	99
	IMGLOAD	Loads image data to VRAM.	Special	Command	99
	IMGSAVE	Saves image data from VRAM.	Special	Command	100
	INKEY\$	Determines the character input from the keyboard.	Key control	Function	100
	INPUT	Assigns data to a variable.	Key control	Command	101
	INPUT#	Reads data from a file.	File control	Command	101
	INPUT\$	Reads a specified length of data.	File control	Function	102
	INPUT WAIT	Inputs data with a time limitation.	Key control	Command	102
	INSTR	Determines the position of a specified character.	1-byte character string operation	Function	103
	INT	Rounds down a value to an integer.	Calculation	Function	103
	INTR ON/OFF/STOP	Disables, enables, or stops interrupts with the STEP signal.	I/O port I/O control	Command	104
	IPL	Sets the OVL boot-up mode.	Special	Command	104
J	JIS\$	Determines the Shift JIS code.	Japanese character string operation	Function	105
K	KACNV\$	Converts from wide to standard characters.	Japanese character string operation	Function	105
	KEXT\$	Selects a character.	Japanese character string operation	Function	105
	KEY	Assigns a character string to function keys.	Key control	Command	106
	KEY LIST	Displays the function key settings.	Program editing	Command	106
	KEY ON/OFF/STOP	Disables, enables, or stops interrupts from the function keys.	Key control	Command	106
	KEYIN	Reads the status of the console keys.	Key control	Function	107
	KILL	Deletes a file.	File operation	Command	107
	KINPUT	Input data in the Japanese input mode.	Key control	Command	108
	KINSTR	Determines the position of a 2-byte character.	Japanese character string operation	Function	108
	KLEN	Determines the length of a character string including 2-byte characters.	Japanese character string operation	Function	108
	KMID\$	Extracts part of a character string including 2-byte characters.	Japanese character string operation	Function	109
	KNJ\$	Determines a 2-byte character.	Japanese character string operation	Function	109

Command/Function		Operation	Instruction	Type	Page
K	KPLOAD	Registers a 2-byte character pattern.	Japanese character string operation	Command	110
	KPOS	Determines the position of a character in a character string.	Japanese character string operation	Function	110
	KTYPE	Determines the type of a character.	Japanese character string operation	Function	111
L	LABEL	Carries out labelling.	Measurement	Command	111
	LBOUND	Determines the lower boundary of an array dimension qualifier.	General	Function	112
	LCASE\$	Converts uppercase letters to-lower case letters.	1-byte character string operation	Function	112
	LDATA	Measures data for the labelled image.	Measurement	Function	112
	LEFT\$	Extracts the left end of the specified character string.	1-byte character string operation	Function	113
	LEN	Determines the length of a character string.	1-byte character string operation	Function	113
	LET	Assigns an expression to a variable.	General	Command	113
	LEVEL	Sets the binary level.	Image processing	Command	114
	LINE	Draws a straight line.	Graphic control	Command	114
	LINE INPUT	Assigns a line of data to a character variable.	Key control	Command	115
	LINE INPUT WAIT	Inputs data with a time limitation.	Key control	Command	116
	LINE INPUT#	Reads data from a file.	File control	Command	116
	LIST	Displays the program contents.	Program editing	Command	116
	LNUM	Determines the number of labelled images.	Measurement	Function	117
	LOAD	Loads a program.	Program editing	Command	117
	LOC	Determines the current position in a specified file.	File control	Function	117
	LOCATE	Sets the cursor position.	Text display control	Command	118
	LOF	Determines the size of a file.	File control	Function	118
	LOG	Determines natural logarithms.	Calculation	Function	118
	LPOINT	Determines the label number at a specified position.	Measurement	Function	119
LPUTIMG	Draws a labelled image.	Measurement	Command	119	
LSET	Writes left-justified character data.	File control	Command	120	
LSORT	Renumbers labels in order of area.	Measurement	Command	120	
LTRIM\$	Deletes spaces to the left of a character string.	1-byte character string operation	Function	120	
M	MASKBIT	Disables writing to specific planes.	Special graphic display control	Command	120
	MDATA	Reads the measured results 1.	Measurement	Function	121
	MDATA2	Reads the measured results 1.	Measurement	Function	121
	MEASURE	Conducts measurements.	Measurement	Command	122
	MENU	Returns to the Menu mode.	Special	Command	122
	MERGE	Merges a program.	Program editing	Command	122

Command/Function	Operation	Instruction	Type	Page	
M	MID\$	Extracts part of a character string.	1-byte character string operation	Function, Command	123
	MKD\$	Converts a double-precision value to a character string.	1-byte character string operation	Function	123
	MKDIR	Creates a new directory.	File operation	Command	124
	MKI\$	Converts an integer to a character string.	1-byte character string operation	Function	124
	MKL\$	Converts a long integer to a character string.	1-byte character string operation	Function	124
	MKS\$	Converts a single-precision value to a character string.	1-byte character string operation	Function	125
	MMODE	Sets the measurement mode.	Measurement	Command	125
N	NAME	Renames files.	File operations	Command	126
	NEW	Deletes a program from memory.	Program execution control	Command	126
	NPIECE	Determines the number of smaller character strings.	1-byte character string operation	Function	126
O	OCT\$	Converts a number to an octal character string.	1-byte character string operation	Function	127
	ON COM GO-SUB	Defines the RS-232C interrupt jump destination.	RS-232C communication	Command	127
	ON ERROR GOTO	Defines the error processing routine.	Error control	Command	128
	ON HELP GO-SUB	Defines the HELP Key interrupt subroutine.	Key control	Command	128
	ON INTR GO-SUB	Defines the STEP signal interrupt subroutine.	I/O port I/O control	Command	128
	ON KEY GO-SUB	Defines the function key interrupt subroutine.	Key control	Command	129
	ON STOP GO-SUB	Defines the STOP Key interrupt subroutine.	Key control	Command	129
	ON TIME\$ GO-SUB	Defines the timer interrupt subroutine.	Time/date control	Command	130
	ON. . . GOSUB	Branches to a subroutine on a specified value.	General	Command	130
	ON. . . GOTO	Jumps program operation on a specified value.	General	Command	131
	OPEN (1)	Opens a file.	File control	Command	131
	OPEN (2)	Opens the RS-232C port.	RS-232C communication	Command	132
OPTION BASE	Declares minimum value of the array qualifier.	General	Command	132	
P	PIECE\$	Returns partial character strings.	1-byte character string operation	Function	133
	PIN	Reads the bit status of an input port.	I/O port I/O control	Function	133
	POINT	Determines the density at specified coordinates.	Graphic control	Function	133
	POLYGON	Draws a polygon.	Graphic control	Command	134
	POLYLINE	Draws a polyline.	Graphic control	Command	134
	POS	Determines the current cursor column position.	Text display control	Function	135
	POUT	Controls the bit status of an output port.	I/O port I/O control	Command	135

Command/Function	Operation	Instruction	Type	Page	
P	PRINT	Displays data on the text display.	Text display control	Command	135
	PRINT USING	Displays formatted data on the text display.	Text display control	Command	136
	PRINT#	Writes data to a sequential access file.	File control	Command	137
	PRINT# USING	Writes formatted data to a sequential access file.	File control	Command	138
	PSET	Draws a point.	Graphic control	Command	138
	PUT#	Writes data to a random access file.	File control	Command	139
	PUT@	Draws a pattern in VRAM.	Graphic control	Command	139
R	RANDOMIZE	Initializes random number generation.	General	Command	140
	RDATA	Reads the detailed run length data.	Measurement	Function	140
	READ	Reads data.	General	Command	140
	REM	Inserts remarks into the program.	General	Command	141
	RENUM	Renums program lines.	Program editing	Command	141
	REPLACE	Replaces a character string.	Program editing	Command	141
	RESTORE	Specifies the line with the DATA statement.	General	Command	142
	RESUME	Restarts the operation before the error occurred.	Error control	Command	142
	RETURN	Returns operation from a subroutine.	General	Command	142
	RIGHT\$	Extracts the right end of the specified character string.	1-byte character string operation	Function	143
	RMDIR	Deletes a directory.	File operation	Command	143
	RMODE	Sets the measurement mode for detailed run length data.	Measurement	Command	144
	RND	Generates a random number.	Calculation	Function	144
	RSET	Writes right-justified character data.	File control	Command	145
	RTRIM\$	Deletes spaces to the right of a character string.	1-byte character string operation	Function	145
	RUN	Runs a program.	Program execution control	Command	145
RUNL	Reads the simple run length data.	Measurement	Function	146	
S	SAVE	Saves a program.	Program editing	Command	146
	SBANK	Selects the shading memory bank number.	Special graphic display control	Command	146
	SCAN	Conducts scan measurement.	Measurement	Command	147
	SCANSET	Sets the conditions for scan measurement.	Measurement	Command	147
	SCNCALIB	Sets the calibration data.	Scene	Command	148
	SCNCAM	Reads the camera data for a scene.	Scene	Function	148
	SCNLEVEL	Determines the binary level of a scene.	Scene layout problem	Function	148
	SCNLOAD	Loads the scene data.	Scene	Command	149
	SCNLUT	Sets the binary level of a scene.	Scene	Command	149
	SCNSAVE	Saves the scene data.	Scene	Command	149
	SDATA 1	Reads the scan measurement data 1.	Measurement	Function	149
	SDATA 2	Reads the scan measurement data 2.	Measurement	Function	150

Command/Function	Operation	Instruction	Type	Page
SEARCH	Determines the number of an element in an array.	General	Function	150
SELECT... CASE ~ CASEELSE ~ END SELECT	Provides multiple branching depending on a value.	General	Command	151
SET	Sets the write-protect attribute for a file.	File operation	Command	151
SETBLUT	Sets array data as binary LUT data.	Image processing	Command	152
SETDLUT	Sets the display LUT data.	Display control	Command	152
SETDLVL	Sets the display level for each display image.	Display control	Command	153
SETLUT	Sets array data as the filter LUT data.	Image processing	Command	153
SFTBLUT	Shifts the binary LUT level.	Image processing	Command	153
SFTLUT	Shifts the filter LUT.	Image processing	Command	154
SGN	Determines the sign of a numeric expression.	Calculation	Function	155
SIN	Determines the sine of a numeric expression.	Calculation	Function	155
SPACE\$	Inserts space characters.	1-byte character string operation/key control	Function	156
SPC	Outputs space characters.	Text display control/key control	Function	156
SPCLOSE	Draws a region bounded by a spline curve.	Graphic control	Command	157
SPLINE	Draws a spline curve.	Graphic control	Command	157
SQR	Determines the square-root.	Calculation	Function	158
SSCROLL	Scrolls the shading master memory.	Special graphic display control	Command	158
STOP	Stops program execution.	General	Command	158
STOP ON/OFF/STOP	Disables, enables, or stops interrupts from the STOP Key.	Key control	Command	159
STR\$	Converts a number to character string representation.	1-byte character string operation	Function	159
STRCHK	Checks for incorrect strobe flashing.	Camera/strobe control	Function	160
STRING\$	Creates a character string with a repeated character.	1-byte character string operation	Function	160
STRMODE	Enables and disables strobe flashing.	Camera/strobe control	Command	160
SUB ~ END SUB	Defines a structural subroutine.	General	Command	161
SWAP	Switches two variables.	General	Command	161
TAB	Specifies the position to display characters.	Text display control	Function	161
TAN	Determines the tangent of a numeric expression.	Calculation	Function	162
TIME\$	Displays and sets the time.	Time/date control	Function, Command	162
TIME\$ ON/OFF/STOP	Disables, enables, or stops timer interrupts.	Time/date control	Command	163
TIMER	Reads and sets the 10 ms timer.	Time/date control	Function, Command	163
TROFF	Exits the Trace mode.	Program execution control	Command	164



Command/Function		Operation	Instruction	Type	Page
T	TRON	Enters the Trace mode.	Program execution control	Command	164
U	UBOUND	Determines the upper boundary of an array dimension qualifier.	General	Function	164
	UCASE\$	Converts lowercase letters to uppercase letters.	1-byte character string operation	Function	164
V	VAL	Converts a character string to a number.	1-byte character string operation	Function	165
	VDWAIT	Delays the VD interrupt the specified number of times.	Special graphic display control	Command	165
	VIDEOIN	Inputs an image.	Camera/strobe control	Command	165
W	WDISP	Sets the type of image display in a window.	Display control	Command	166
	WHILE ~ WEND	Repeatedly executes instructions.	General	Command	166
	WINDOW	Draws a window.	Scene	Command	167
	WRITE	Displays data on the display.	Text display control	Command	167
	WRITE#	Writes data to a sequential file.	File control	Command	168
	WSCROLL	Scrolls the window memory.	Special graphic display control	Command	168

## 1-2 OVL Instruction List

Instruction	Command/Function	Operation	Page	
Program editing	Command	AUTO	Automatically generates line numbers.	56
		DELETE	Deletes part of a program.	76
		EDIT	Selects the mode to edit lines of the program.	81
		FIND	Searches for a character string in a program.	89
		HELP	Displays help messages.	96
		KEY LIST	Displays the function key settings.	106
		LIST	Displays the program contents.	116
		LOAD	Loads a program.	117
		MERGE	Merges a program.	122
		RENUM	Renums program lines.	141
		REPLACE	Replaces a character string.	141
		SAVE	Saves a program.	146
Program execution control	Command	CONT	Continues execution of a program.	68
		NEW	Deletes a program from memory.	126
		RUN	Runs a program.	145
		TROFF	Exits the Trace mode.	164
		TRON	Enters the Trace mode.	164
File operation	Command	CHDIR	Changes the current directory.	62
		FILES	Displays file names.	88
		KILL	Deletes a file.	107
		MKDIR	Creates a new directory.	124
		NAME	Renames files.	126
		RMDIR	Deletes a directory.	143
		SET	Sets the write-protect attribute for a file.	151
General	Command	CALL	Calls a structural subroutine.	60
		CHAIN	Transfers control to a specified program.	61
		COMMON	Transfers variables.	67
		DATA	Defines the data read with the READ instruction.	72
		DEF FN	Defines a user function.	73
		DEF FN ~ END DEF	Defines a user function block.	74
		DEFDBL	Declares variables as double-precision real number variables.	74
		DEFINT	Declares variables as integers.	74
		DEFLNG	Declares variables as long integers.	75
		DEFSNG	Declares variables as single-precision real number variables.	75
		DEFSTR	Declares variables as character variables.	75
		DIM	Defines an array variable.	76
		DO . . . LOOP REPEAT	Repeats a loop specified number of times.	78
		DO . . . LOOP UNTIL	Repeats a loop until a condition is fulfilled.	78

Instruction	Command/Function	Operation	Page		
General	Command	DO... LOOP WHILE	Repeats a loop while a condition is fulfilled.	79	
		DO REPEAT LOOP	Repeats a loop specified number of times.	79	
		DO UNTIL LOOP	Repeats a loop until a condition is fulfilled.	80	
		DO WHILE LOOP	Repeats a loop while a condition is fulfilled.	80	
		END	Stops program execution.	82	
		ERASE	Deletes an array variable.	83	
		EXIT DEF/DO/FOR/SUB	Exits a control block.	86	
		FOR... TO... STEP ~ NEXT	Repeatedly executes instructions.	91	
		GOSUB	Branches to a specified subroutine.	95	
		GOTO/GO TO	Unconditionally jumps to a specified line.	96	
		IF... GOTO ~ ELSE	Evaluates a condition.	98	
		IF... THEN ~ ELSE	Evaluates a condition.	98	
		IF... THEN ~ ELSEIF ~ ELSE ~ END IF	Evaluates a condition.	99	
		LET	Assigns an expression to a variable.	113	
		ON... GOSUB	Branches to a subroutine on a specified value.	130	
		ON... GOTO	Jumps program operation on a specified value.	131	
		OPTION BASE	Declares minimum value of the array qualifier.	132	
		RANDOMIZE	Initializes random number generation.	140	
		READ	Reads data.	140	
		REM	Inserts remarks into the program.	141	
		RESTORE	Specifies the line with the DATA statement.	142	
		RETURN	Returns operation from a subroutine.	142	
		SELECT... CASE ~ CASEELSE ~ END SELECT	Provides multiple branching depending on a value.	151	
		STOP	Stops program execution.	158	
		SUB ~ END SUB	Defines a structural subroutine.	161	
		SWAP	Switches two variables.	161	
		WHILE ~ WEND	Key control	166	
			Function	DATE\$	Displays and sets the date.
		LBOUND		Determines the lower boundary of an array dimension qualifier.	112
		SEARCH		Determines the number of an element in an array.	150
	UBOUND	Determines the upper boundary of an array dimension qualifier.		164	
Text display	Command	COLOR	Changes the character attributes.	66	
		COLOR@	Changes the character attributes within a specified region.	66	
		CONSOLE	Sets the text display mode.	68	

Instruction		Command/Function	Operation	Page
Text display	Command	LOCATE	Sets the cursor position.	118
		PRINT	Displays data on the text display.	135
		PRINT USING	Displays formatted data on the text display.	136
		WRITE	Camera/strobe control	167
	Function	CSRLIN	Determines the cursor line position.	69
		POS	Determines the current cursor column position.	135
		SPC	Outputs space characters.	156
		TAB	Specifies the position to display characters.	161
Graphic control	Command	ARC	Draws an arc.	54
		BOX	Draws a rectangle.	59
		CIRCLE	Draws a circle.	64
		CLS	Clears characters and graphics.	66
		CURSOR	Draws a cross cursor.	69
		ELLIPSE	Draws an ellipse.	82
		GET@	Reads image data to VRAM.	93
		LINE	Draws a straight line.	114
		POLYGON	Draws a polygon.	134
		POLYLINE	Draws a polyline.	134
		PSET	Draws a point.	138
		PUT@	Draws a pattern in VRAM.	139
		SPCLOSE	Draws a region bounded by a spline curve.	157
		SPLINE	Draws a spline curve.	157
	Function	POINT	Determines the density at specified coordinates.	133
Special graphic display control	Command	BCOPY	Copies a binary image.	57
		BCOPY2	Makes enlarged and reduced copies of binary images.	58
		GCOPY	Copies the raw image in VRAM.	92
		GCOPY2	Makes enlarged and reduced copies of the raw image.	92
		MASKBIT	Disables writing to specific planes.	120
		SBANK	Selects the shading memory bank number.	146
		SSCROLL	Scrolls the shading master memory.	158
		VDWAIT	Delays the VD interrupt the specified number of times.	165
		WSCROLL	Camera/strobe control	168
Calculation	Function	ABS	Determines an absolute value.	54
		ATN	Determines the arctangent (arctan) of a value.	55
		CDBL	Converts to a double-precision value.	61
		CINT	Converts to an integer.	63

Instruction		Command/Function	Operation	Page
Calculation	Function	CLNG	Converts to a long integer.	65
		COS	Determines the cosine of an angle.	69
		CSNG	Converts to a single-precision value.	69
		CVD	Converts an 8-byte character string to a double-precision real number.	70
		CVI	Converts a 2-byte character string to an integer.	71
		CVL	Converts a 4-byte character string to a long integer.	71
		CVS	Converts a 4-byte character string to a single-precision real number.	71
		EXP	Determines the natural number e raised to an exponential power.	87
		FIX	Rounds down a value to an integer.	90
		INT	Rounds down a value to an integer.	103
		LOG	Determines natural logarithms.	118
		RND	Generates a random number.	144
		SGN	Determines the sign of a numeric expression.	155
		SIN	Determines the sine of a numeric expression.	155
		SQR	Determines the square-root.	158
		TAN	Determines the tangent of a numeric expression.	162
Error control	Command	ERRMSG	Defines the operation when an error occurs.	84
		ERROR	Generates a pseudo-error.	84
		ON ERROR GOTO	Defines the error processing routine.	128
		RESUME	Restarts the operation before the error occurred.	142
	Function	ERL	Determines the line number where an error occurred.	84
		ERR	Determines the error code.	84
1-byte character string operation	Command	MID\$	Extracts part of a character string.	123
	Function	ASC	Determines the character code.	55
		CHR\$	Converts a character code to a character.	63
		HEX\$	Converts a numeric expression to hexadecimal.	97
		INSTR	Determines the position of a specified character.	103
		LCASE\$	Converts uppercase letters to-lower case letters.	112
		LEFT\$	Extracts the left end of the specified character string.	113
		LEN	Determines the length of a character string.	113
		LTRIM\$	Deletes spaces to the left of a character string.	120
		MID\$	Extracts part of a character string.	123

Instruction		Command/Function	Operation	Page
1-byte character string operation	Function	MKD\$	Converts a double-precision value to a character string.	123
		MKI\$	Converts an integer to a character string.	124
		MKL\$	Converts a long integer to a character string.	124
		MKS\$	Converts a single-precision value to a character string.	125
		NPIECE	Determines the number of smaller character strings.	126
		OCT\$	Converts a number to an octal character string.	127
		PIECE\$	Returns partial character strings.	133
		RIGHT\$	Extracts the right end of the specified character string.	143
		RTRIM\$	Deletes spaces to the right of a character string.	145
		SPACE\$	Inserts space characters.	156
		STR\$	Converts a number to character string representation.	159
		STRING\$	Creates a character string with a repeated character.	160
		UCASE\$	Converts lowercase letters to uppercase letters.	164
		VAL	Converts a character string to a number.	165
Japanese character string operation	Command	KPLOAD	Registers a 2-byte character pattern.	110
	Function	AKCNV\$	Converts 1-byte characters to 2-byte characters.	54
		JIS\$	Determines the Shift JIS code.	105
		KACNV\$	Converts from wide to standard characters.	105
		KEXT\$	Selects a character.	105
		KINSTR	Determines the position of a 2-byte character.	108
		KLEN	Determines the length of a character string including 2-byte characters.	108
		KMID\$	Extracts part of a character string including 2-byte characters.	109
		KNJ\$	Determines a 2-byte character.	109
		KPOS	Determines the position of a character in a character string.	110
		KTYPE	Determines the type of a character.	111
File control	Command	CLOSE	Closes an open file.	65
		FIELD#	Allocates the variable areas.	87
		GET#	Reads data from a random file.	93
		INPUT#	Reads data from a file.	101
		LINE INPUT#	Reads data from a file.	116
		LSET	Writes left-justified character data.	120
		OPEN (1)	Opens a file.	131
		PRINT#	Writes data to a sequential access file.	137

Instruction		Command/Function	Operation	Page		
File control	Command	PRINT# USING	Writes formatted data to a sequential access file.	138		
		PUT#	Writes data to a random access file.	139		
		RSET	Writes right-justified character data.	145		
		WRITE#	1-byte character string operation	168		
	Function	ATTR\$	Determines the write protect attribute of a file.	55		
		DSKF	Determines the free space in the memory card.	81		
		EOF	Determines the end of the file.	83		
		INPUT\$	Reads a specified length of data.	102		
		LOC	Determines the current position in a specified file.	117		
		LOF	Determines the size of a file.	118		
Key control	Command	HELP ON/OFF/STOP	Disables, enables, or stops interrupts from the HELP Key.	96		
		INPUT	Assigns data to a variable.	101		
		INPUT WAIT	Inputs data with a time limitation.	102		
		KEY	Assigns a character string to function keys.	106		
		KEY ON/OFF/STOP	Disables, enables, or stops interrupts from the function keys.	106		
		KINPUT	Input data in the Japanese input mode.	108		
		LINE INPUT	Assigns a line of data to a character variable.	115		
		LINE INPUT WAIT	Inputs data with a time limitation.	116		
		ON HELP GOSUB	Defines the HELP Key interrupt subroutine.	128		
		ON KEY GOSUB	Defines the function key interrupt subroutine.	129		
		ON STOP GOSUB	Defines the STOP Key interrupt subroutine.	129		
		STOP ON/OFF/STOP	Disables, enables, or stops interrupts from the STOP Key.	159		
		Function	INKEY\$	Determines the character input from the keyboard.	100	
	KEYIN		Reads the status of the console keys.	107		
	SPACE\$		Inserts space characters.	156		
	SPC		Outputs space characters.	156		
	Time/date control		Command	DATE\$	Displays and sets the date.	73
				ON TIME\$ GOSUB	Defines the timer interrupt subroutine.	130
		TIME\$		Displays and sets the time.	162	
TIME\$ ON/OFF/STOP		Disables, enables, or stops timer interrupts.		163		
TIMER		Reads and sets the 10 ms timer.		163		
Function		DATE\$	Displays and sets the date.	73		
		TIME\$	Displays and sets the time.	162		
	TIMER	Reads and sets the 10 ms timer.	163			

Instruction	Command/Function		Operation	Page
RS-232C communication	Command	COM ON/OFF/STOP	Controls interrupts from the RS-232C.	67
		ON COM GOSUB	Defines the RS-232C interrupt jump destination.	127
		OPEN (2)	Opens the RS-232C port.	132
I/O port I/O control	Command	BUSY	Controls BUSY signal output status.	59
		DOUT	Outputs data to an output port.	80
		ERRROUT	Controls the error signal output.	85
		GATE	Controls the GATE signal.	91
		INTR ON/OFF/STOP	Disables, enables, or stops interrupts with the STEP signal.	104
		ON INTR GOSUB	Defines the STEP signal interrupt subroutine.	128
	POUT	Controls the bit status of an output port.	135	
	Function	DIN	Reads data from an input port.	77
		DSA	Reads the status of the DSA signal.	81
PIN		Reads the bit status of an input port.	133	
Memory management	Command	CLEAR	Initializes variables.	65
	Function	FRE	Determines the free memory size.	91
Special	Command	BEEP	Turns the buzzer on and off.	58
		DEVICE	Specifies the input and output devices.	76
		IMGLOAD	Loads image data to VRAM.	99
		IMGSAVE	Saves image data from VRAM.	100
		IPL	Sets the OVL boot-up mode.	104
		MENU	Returns to the Menu mode.	122
Scene	Command	CHANGE	Changes the measuring conditions.	62
		SCNCALIB	Sets the calibration data.	148
		SCNLOAD	Loads the scene data.	149
		SCNLUT	Sets the binary level of a scene.	149
		SCNSAVE	Saves the scene data.	149
		WINDOW	1-byte character string operation	167
	Function	SCNCAM	Reads the camera data for a scene.	148
		SCNLEVEL	Determines the binary level of a scene.	148
Measurement	Command	LABEL	Carries out labelling.	111
		LPUTIMG	Draws a labelled image.	119
		LSORT	Renumbers labels in order of area.	120
		MEASURE	Conducts measurements.	122
		MMODE	Sets the measurement mode.	125
		RMODE	Sets the measurement mode for detailed run length data.	144
		SCAN	Conducts scan measurement.	147
		SCANSET	Sets the conditions for scan measurement.	147
	Function	LDATA	Measures data for the labelled image.	112
		LNUM	Determines the number of labelled images.	117



Instruction		Command/Function	Operation	Page
Measurement	Function	LPOINT	Determines the label number at a specified position.	119
		MDATA	Reads the measured results 1.	121
		MDATA2	Reads the measured results 1.	121
		RDATA	Reads the detailed run length data.	140
		RUNL	Reads the simple run length data.	146
		SDATA 1	Reads the scan measurement data 1.	149
		SDATA 2	Reads the scan measurement data 2.	150
Image processing	Command	ENHANCE	Creates LUT data for contrast modification.	83
		FILTDATA	Specifies the line filter factors.	88
		FILTER	Sets the image filtering function.	88
		FILTERIN	Selects image input to the filter.	89
		GETBLUT	Reads the binary LUT data.	94
		GETLUT	Reads the filter LUT data.	95
		HISTGRAM	Reads a density histogram.	97
		LEVEL	Sets the binary level.	114
		SETBLUT	Sets array data as binary LUT data.	152
		SETLUT	Sets array data as the filter LUT data.	153
		SFTBLUT	Shifts the binary LUT level.	153
		SFTLUT	Shifts the filter LUT.	154
		Function	AUTOLVL	Determines the binary level.
	Camera/strobe control	Command	CAMERA	Switches cameras.
CAMMODE			Selects the image output from a Camera I/F Unit.	60
EVENTIN			Sets the image input synchronized with events.	85
FLASH			Controls the strobe and shutter function.	90
STRMODE			Enables and disables strobe flashing.	160
VIDEOIN			Inputs an image.	165
Function		CAMCHK	Determines the camera connection status.	60
		STRCHK	Checks for incorrect strobe flashing.	160
Display control	Command	BACKDISP	Sets the image display outside the window.	56
		DISPLAY	Sets the display image.	77
		GETDLUT	Reads the display LUT data.	94
		SETDLUT	Sets the display LUT data.	152
		SETDLVL	Sets the display level for each display image.	153
		WDISP	General	166
	Function	GETDLVL	Determines the display level of the image.	95

## SECTION 2

### OVL Syntax

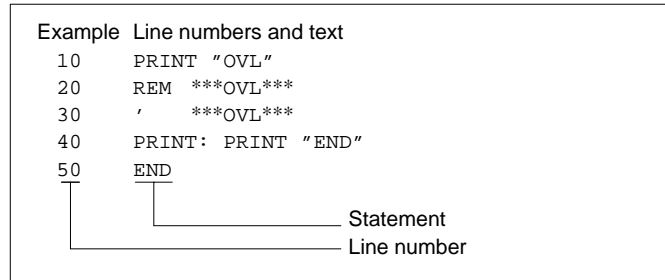
This section provides the basic OVL syntax required before programming.

2-1	Line Format .....	20
2-1-1	Lines .....	20
2-1-2	Line Numbers .....	20
2-1-3	Statements .....	20
2-2	OVL Characters and Symbols .....	20
2-3	Constants .....	21
2-3-1	Character Constants .....	21
2-3-2	Numeric Constants .....	22
2-4	Variables .....	23
2-4-1	Type Declarators .....	23
2-4-2	Variable Names .....	24
2-4-3	Reserved Words .....	24
2-4-4	Array Variables .....	24
2-5	Type Conversion .....	25
2-6	Operations and Expressions .....	27
2-6-1	Operators .....	27
2-6-2	Expressions .....	29
2-6-3	Priority of Operations .....	32
2-7	Interrupts .....	32
2-8	Labels .....	32

## 2-1 Line Format

### 2-1-1 Lines

An OVL program is composed of lines.



Each line consists of a line number and a statement. Each line, including the line number, spaces, and the instruction may be up to 255 bytes in length.

Normally a line declares a single statement but it is possible to include multiple statements in a line. Multiple statements in a single line are delimited by colons (:).

### 2-1-2 Line Numbers

The line numbers are positioned at the start of a line and are integers applied in ascending order between 1 and 65535. If line numbers are not applied in ascending order, the lines are automatically rearranged into ascending order. Line numbers may start and finish with any number between 1 and 65535.

Lines assigned with line numbers are stored as part of the program when the Return Key is pressed. A line with no line number is not stored as part of the program but is executed immediately when the Return Key is pressed.

The program is executed in the order of the line numbers, except where branching occurs.

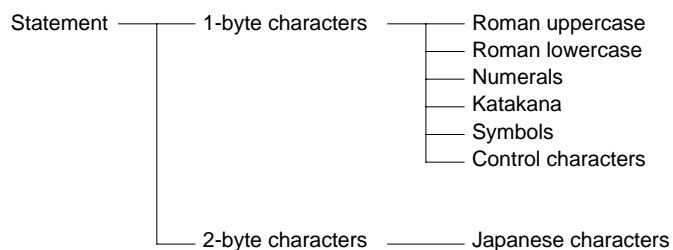
### 2-1-3 Statements

A statement is the smallest unit used to declare an OVL procedure.

Statements include executable statements which declare and execute OVL commands and functions, non-executable statements which provide program comments, and labels which define jump destinations.

## 2-2 OVL Characters and Symbols

The following characters and symbols can be used with OVL programming.



Only lowercase characters defined in a character constant or character variable inside quotes (" ") are handled as lowercase characters. The system automatically converts other lowercase characters to uppercase characters.

### Symbols

#### Period (.)

The period indicates the current line number (the last line number executed). It can replace the line number in the following commands: AUTO, EDIT, LIST, LLIST, RENUM.

Minus (-)

The minus specifies a range of lines.

Colon (:)

The colon delimits one statement from another.

Comma (,)

The comma delimits parameters in a series.

Semicolon (;)

The semicolon delimits parameters in an output statement.

Apostrophe (')

Identical in meaning to a REM statement.

Question Mark (?)

Simplifies input of the word "PRINT." Immediately after input, the question mark is converted to "PRINT."

Asterisk (\*)

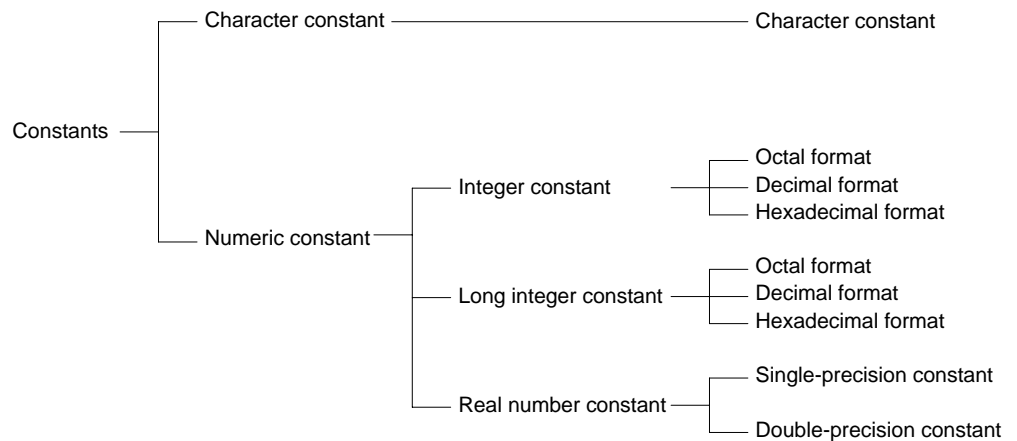
The asterisk indicates the start of a label name.

Space

A space must be inserted between a command and the following parameter. Other spaces may be inserted anywhere except inside command names, variable names, or numeric values.

## 2-3 Constants

Values or character strings declared directly in the program are known as constants. A character constant is declared differently from a numeric constant.



### 2-3-1 Character Constants

A character constant is a character string enclosed in double quotation marks (") and can contain up to 255 characters.

The CHR\$ function must be used to include double quotations inside a character string, as shown in one of the examples below.

A character string of zero length is known as a "null string."

#### Character Strings

"1234567890" . . . . Arithmetic operations cannot be carried out on numbers in this form.

"NEW PLAN" . . . . This character string contains 8 characters, 7 letters and one space.

CHR\$(34) . . . . . This character string represents a single double quotation (") character.

"" ..... This is a null string.

## 2-3-2 Numeric Constants

A numeric constant directly declares a value for operation by the arithmetic operators.

The plus (+) or minus (–) sign can be included before the value to indicate a positive or negative value, but the plus sign may be omitted.

Numeric constants can be declared as integers or real numbers.

### Integer Constants

#### Decimal Format

Integers must lie in the range –32768 to +32767.

Adding a % sign to a real number in the above range rounds off the value to an integer constant.

Examples of decimal format integer constants:

32767

–123

+5%

321.01% (This is identical to 321.)

#### Octal Format

An octal integer constant is declared as an octal value (digits 0 to 7) preceded by the prefix &O. However, the “O” can be omitted if required.

Octal integer constants can be specified between &O0 and &O177777.

Examples of octal format integer constants:

&12345

&O77777

#### Hexadecimal Format

A hexadecimal integer constant is declared as a hexadecimal value (digits 0 to 9, A to F) preceded by the prefix &H.

Hexadecimal integer constants can be specified between &H0 and &HFFFF.

Examples of hexadecimal format integer constants:

&H100

&HFCA0

### Long Integer Constants

#### Decimal Format

Integers must lie in the range –2147483648 to +2147483647.

Adding a & sign to a real number in the above range rounds off the value to a long integer constant.

Examples of decimal format long integer constants:

9500000&

–365432&

100.12& (This is identical to 100.)

#### Octal Format

An octal integer constant is declared as an octal value (digits 0 to 7) preceded by the prefix &O and followed by the type declarator (&).

Octal integer constants can be specified between &O0& and &O37777777777&.

Examples of octal format long integer constants:

&O12345&

&O112377777777&

Hexadecimal Format

A hexadecimal integer constant is declared as a hexadecimal value (digits 0 to 9, A to F) preceded by the prefix &H and followed by the type declarator (&).

Hexadecimal integer constants can be specified between &H0& and &HFFFFFFF&.

Examples of hexadecimal format long integer constants:

```
&H100&
&HFCA00&
```

**Note** Values input in octal and hexadecimal format are printed out in decimal format.

**Real Number Constants**Single-precision Format

Real number data between  $-1.70141E+38$  to  $+1.70141E+38$  to 6 significant digits.

The following numbers are declared as single-precision real numbers:

- Values with a decimal point but not exceeding 6 significant digits
- Values with the single-precision declarator (!)
- Values in the single-precision E exponent format.

Examples of single-precision constants:

```
108.12
3.14!
-8.76E-6
```

Double-precision Format

Real number data between  $-1.701411834604692D+38$  to  $+1.701411834604692D+38$  to 16 significant digits.

The following numbers are declared as double-precision real numbers:

- Real numbers with 7 or more digits
- Values with the double-precision declarator (#)
- Values in the double-precision D exponent format.

Examples of double-precision constants:

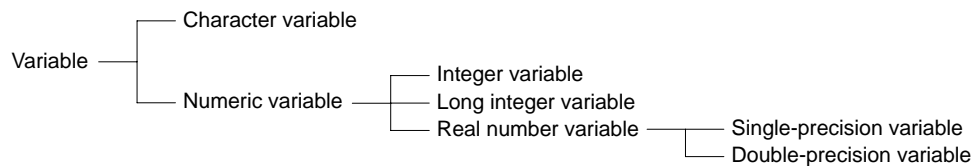
```
108.123456
3.14#
-8.76D-6
```

## 2-4 Variables

Named areas in the program in which values or character strings are stored are known as variables.

Before a value is allocated, a numeric variable contains 0 and a character variable contains a null string ("").

The user can declare both the name and type of a variable.



### 2-4-1 Type Declarators

The type of variable is declared with a declarator suffix to the variable name.

Variables with an identical variable name but different declarators are treated as separate variables.

Types of Declarator:

\$ . . . . Character  
% . . . . Integer  
& . . . . Long integer  
! . . . . Single-precision  
# . . . . Double-precision

The type of variable can also be declared with a type statement. However, the type declared with the type declarator takes priority over the type statement.

Types of Type Statement:

DEFSTR . . . . Declares a character variable.  
DEFINT . . . . Declares an integer variable.  
DEFLNG . . . . Declares a long integer variable.  
DEFSNG . . . . Declares a single-precision variable.  
DEFDBL . . . . Declares a double-precision variable.

Variables not declared with a type declarator or type statement are treated as single-precision numeric variables.

Examples of type statements:

```
DEFSTR NAME$  
DEFDBL AX#, AY#, AZ#
```

The \$ and # can be omitted from these statements.

## 2-4-2 Variable Names

Variable names are limited to 40 characters, including the type declarator. Alphanumeric characters, the period, and type declarator are valid in a variable name.

The name must start with an alphabetic character. Uppercase and lowercase characters are not differentiated. The type declarator must be added to the end of the variable name.

Symbols cannot be used in variable names.

Reserved words cannot be used as variable names. However, variable names may contain reserved words.

Variable names cannot begin with the letters "FN."

## 2-4-3 Reserved Words

Reserved words are character strings defined in the system, such as commands, functions, and operators. Users cannot use reserved words as variable names.

The reserved words are listed in *Appendix B, Reserved Words*.

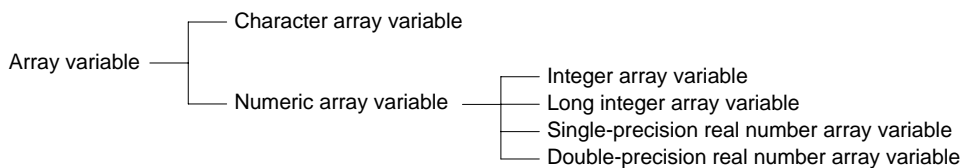
## 2-4-4 Array Variables

A variable with a single variable name storing multiple values is known as an array variable.

Array variables storing character strings are classified as character array variables and array variables storing numeric values are classified as numeric array variables.

Numeric array variables are further subdivided according to the type of numeric values they store into integer array variables, long integer array variables,

single-precision real number array variables, and double-precision real number array variables.



The dimension of the array variable and the subscript range are declared with the DIM statement.

It is unnecessary to use the DIM statement to declare subscripts up to 10.

Example:

*DIM A (10, 10, 10)* may be omitted.

The dimensions of the array must be declared in a single program line (255 characters). The subscripts are limited by the amount of memory. Consequently, 4-dimensional, 5-dimensional, and 6-dimensional arrays may be declared but the number of elements are restricted.

Example:

DIM A(10)	1-dimensional array, number of elements = 11
DIM TA(10, 50)	2-dimensional array, number of elements = 11 x 51 = 561
DIM TTA\$(2, 5, 3)	3-dimensional array, number of elements = 3 x 6 x 4 = 72
. . .	4-dimensional array
. . .	5-dimensional array
. . .	6-dimensional array

Subscripts start from zero, so that the action number of elements is 1 plus the subscript.

## 2-5 Type Conversion

If necessary, the type of numeric data can be converted under the conditions described below.

It is not possible to convert between character and numeric data.

### Condition 1

When data is assigned to a variable of a different type, the data is converted to the type declared with the variable type declarator.

Example of assigning a value to a variable of a different type:

```

10 A%=1.234 . . Assigns the single-precision real number 1.234 to the
                    integer variable A% (assigned as 1).
20 PRINT A% . . Displays the value stored in the integer variable A% on
                    the screen.
  
```

Result:

```

1 . . . . . The integer 1 is displayed.
  
```

### Condition 2

Before operations are carried out on values with different accuracies, all values are first converted to the accuracy of the highest accuracy value.

Example of operations on values of different accuracy:

```

10 A%=10%/3%
20 PRINT A% . . The result of the operation on integers is displayed as
                    an integer.
30 B!=10%/3!
  
```



40 PRINT B! . . . The integer 10% is converted to a single-precision real number, the operation is carried out on single-precision values, and the result displayed as a single-precision value.

50 C#=10%/3#

60 PRINT C# . . . The integer 10% is converted to a double-precision real number, the operation is carried out on double-precision values, and the result displayed as a double-precision value.

Results:

3 . . . . . Result of execution of line 20 is an integer.

3.33333 . . . . . Result of execution of line 40 is a single-precision real number.

3.333333333333333 . . . Result of execution of line 60 is a double-precision real number

### Condition 3

Before logical operations all values are converted into integers and the results are also integers.

Example of logical operations:

10 A=1.234 . . . Assigns the value 1.234 to the numeric variable A.

20 B=NOT A . . . Converts the value 1.234 stored in the numeric variable A to the integer 1, executes the NOT logical operation, and assigns the result (-2) to the numeric variable B.

30 PRINT B, A Displays the values stored in the numeric variables B and A on the screen.

Result:

-2 1.234 . . . . . Result of execution of line 30 is displayed on the screen.

### Condition 4

Real numbers are rounded off to an integer. An error occurs if the converted value lies outside integer range.

Example of conversion to an integer:

10 A%=1.45 . . . Assigns the value 1.45 rounded off to 1 to the integer variable A%.

20 B%=1.65 . . . Assigns the value 1.65 rounded off to 2 to the integer variable B%.

30 PRINT A%,B% Displays the values stored in the integer variables A% and B% on the screen.

Result:

1 2 . . . . . Result of execution of line 30 is displayed on the screen.

### Condition 5

When a double-precision variable is assigned to a single-precision variable, the variable is rounded off to 6 significant digits, with subsequent digits rounded off.

Example of conversion to a single-precision, real number variable:

10 A!=3.14159265358 . Substitutes the effective six digits, 3.14159, for a single-precision, real number variable A!.

20 PRINT A! . . . . . The value to be stored as the single-precision, real number variable A! will be displayed (the seventh digit will be rounded to the nearest whole number).

Result:

3.14159 . . . Result of execution of line 20 is displayed on the screen.

**Note** OVL does not convert between numeric and character data, except for special applications such as random access file I/O and conversion of number declaration character strings to numeric data.

Special functions are used to convert between numeric and character data, where required.

Random access file I/O:

- CVI/CVL/CVS/CVD functions
- MKI\$/MKL\$/MKS\$/MKD\$ functions

Conversion of a number declaration character string to numeric data:

VAL function

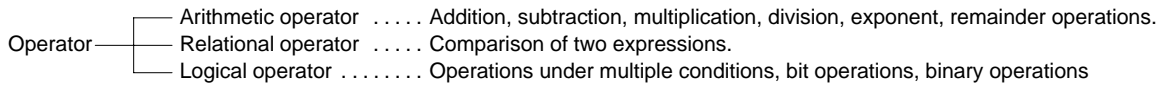
Conversion of numeric data to a character string:

STR\$ function

## 2-6 Operations and Expressions

### 2-6-1 Operators

Three types of operator are used in OVL: arithmetic operators, relational operators, and logical operators.



#### Arithmetic Operators

Arithmetic operators link numeric constants or variables to carry out addition, subtraction, multiplication, division, exponent, and remainder operations.

Arithmetic operator	Description	Declaration example	Mathematical notation
+	Addition	A + B	A + B
-	Subtraction	A - B	A - B
*	Multiplication	A * B	A x B or AB
/	Real number division	A / B	A ÷ B or A/B
¥	Integer division	A ¥ B	[A/B] [ ] indicates Gauss' notation
^	Exponent operation	A ^ B	A <sup>B</sup>
MOD	Remainder calculation	A MOD B	A - [A/B] x B [ ] indicates Gauss' notation

If either the divisor or dividend for integer division is a real number, the real number is rounded off to an integer before division. If the quotient contains decimal places, these are dropped.

Example of integer division:

$$123.4 \text{ ¥ } 67.89 \quad A \quad 123 \text{ ¥ } 68 \quad A \quad 1.808 \quad A \quad 1$$

Decimals rounded off A Division A Decimals dropped A Results

The decimal places are rounded off before the remainder calculation is carried out on real numbers. The result is the remainder from the integer division.

Example of remainder calculation:

$$13.3 \text{ MOD } 4 = 1 \quad \text{Remainder of 13 divided by 4.}$$

$$25.68 \text{ MOD } 6.99 = 5 \quad \text{Remainder of 26 divided by 7.}$$

Digit overflow occurs if the result of a calculation exceeds the range of the value type. If the digits overflow, an error is output and the calculation continues with the maximum value the computer can handle.

Example of digit overflow:

$$A\% = 32760 + 10$$

An error is output and the A% value becomes 32767.

If division by 0 is carried out during operation, an error is output and the calculation continues with the maximum value the computer can handle. The same applies if the exponent operation is carried out on 0 with a negative exponent.

**Relational Operators**

Relational operators compare two numeric data or 2 character data.

True (-1) is returned if the result of the compared data is the same or False (0) is returned if the result of the compared data is different.

Relational operator	Description	Declaration example
=	Equals	A = B
< >, > <	Not equals	A < > B, A > < B
<	Less than	A < B
>	Greater than	A > B
< =, = <	Less than or equal to	A < = B, A = < B
> =, = >	Greater than or equal to	A > = B, A = > B

Relational operators are used inside the IF statement to control the flow of program execution.

Examples of relational operators inside the IF statement:

*IF A<>B THEN 1000 . . . . .* Jump to line 1000 if A is not equal to B (A < > B is true (-1)).

*IF A\$ = "Y" THEN \*PROCESS1* Jump to label \*PROCESS1 if A\$ is equal to " Y" (A\$ = " Y" is true (-1)).

**Logical Operators**

Logical operators are used to investigate multiple conditions and carry out bit operations or binary (Boolean) operations on a specified value.

The handled values are first converted to a two's complement display integers between -32768 to +32767 before the value 0 or 1 is assigned to each bit as the result of the operation. An error occurs if this range is exceeded during conversion.

Logical operator	Description	Equivalence
NOT	Declaration example	NOT A
AND	Negation	A AND B
OR	Logical product	A OR B
XOR	Logical sum	A XOR B
IMP	Exclusive OR	A IMP B
EQV	Inclusive OR	A EQV B

Refer to page 29 for details of the results of logical operations.

## 2-6-2 Expressions

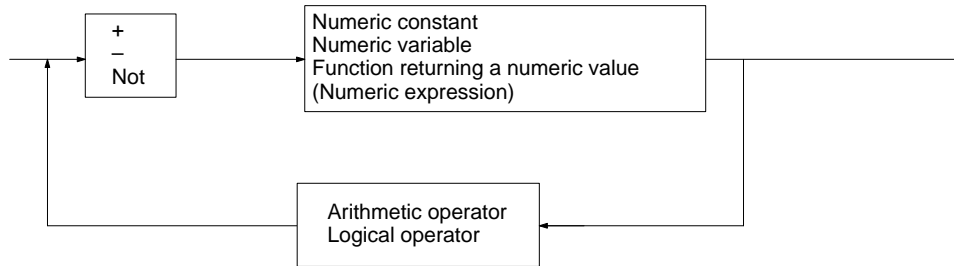
In an OVL program, an “expression” refers to constants, variables, functions, numeric constants and variables linked with arithmetic operators, and character constants and variables linked with plus signs (+).

Examples of expressions:

Numeric expressions	Character expressions	Function expressions	Logical expressions
A=20*15/3 S=P*D*D/4 2.7145 A SIN (X)	A\$="OVL"+"BASIC" A\$=B\$+C\$ "OVL" A\$ CHR\$ (31)	A=B A<B A<>B	A AND B A OR B A XOR B A<B AND A>C A=B OR A<>C

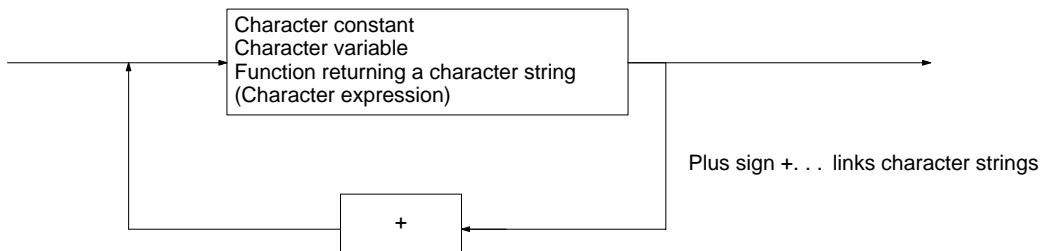
### Numeric Expressions

An expression returning a numeric value is known as a numeric expression. Numeric expressions can be numeric constants, numeric variables, or functions returning numeric values linked by arithmetic or logical operators. Multiple expressions contained within parentheses ( ) can be linked together.



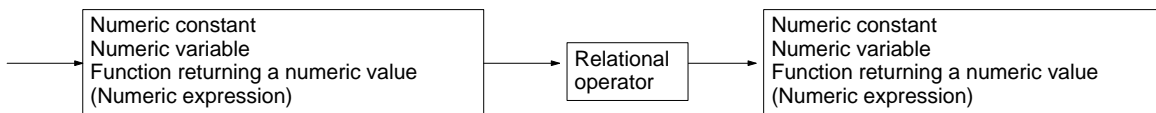
### Character Expressions

An expression returning a character string is known as a character expression. Character expressions can be character constants, character variables, or functions returning character strings linked by plus signs. Multiple expressions contained within parentheses ( ) can be linked together.



### Relational Expressions

A pair of numeric expressions linked by a relational operator is known as a relational expression. Relational expressions can be numeric constants, numeric variables, or functions returning numeric values linked by relational operators.

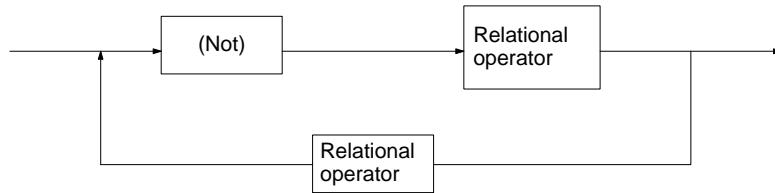


### Logical Expressions

Multiple relational expressions linked by logical operators are known as a logical expression. A logical expression is made up of several relational expressions linked by logical operators to execute bit or binary operations or to evaluate multiple conditions.

Logical expressions have a numeric value and may be used in any position as a numeric expression.

A logical expression is True if the result of the expression is -1 or False if the result is 0.



Logical expressions may include six types of logical operator: NOT, OR, AND, XOR, IMP and EQV.

The results of operation by these logical operators on a single bits A and B are shown in the following tables.

**NOT**

A	NOT A Result
0	1
1	0

**XOR**

A	B	A XOR B Result
0	0	0
0	1	1
1	0	1
1	1	0

**AND**

A	B	A AND B Result
0	0	0
0	1	0
1	0	0
1	1	1

**IMP**

A	B	A IMP B Result
0	0	1
0	1	1
1	0	0
1	1	1

**OR**

A	B	A OR B Result
0	0	0
0	1	1
1	0	1
1	1	1

**EQV**

A	B	A EQV B Result
0	0	1
0	1	0
1	0	0
1	1	1

Examples of bit operations using logical expressions:

Logical expression (NOT): NOT 5      Result: -6

Logical expression	Integer representation	Binary representation
---	5	0000000000000101
NOT 5	-6	111111111111010

Logical expression (AND): 3 AND 5      Result: 1

Logical expression	Integer representation	Binary representation
---	3	0000000000000011
---	5	0000000000000101
3 AND 5	1	0000000000000001

Logical expression (OR): 3 OR 5      Result: 7

Logical expression	Integer representation	Binary representation
---	3	0000000000000011
---	5	0000000000000101
3 OR 5	7	0000000000000111

Logical expression (XOR): 3 XOR 5      Result: 6

Logical expression	Integer representation	Binary representation
---	3	0000000000000011
---	5	0000000000000101
3 XOR 5	6	0000000000000110

Logical expression (IMP): 3 IMP 5      Result: -3

Logical expression	Integer representation	Binary representation
---	3	0000000000000011
---	5	0000000000000101
3 IMP 5	-3	1111111111111101

Logical expression (IMP): 3 EQV 5      Result: -7

Logical expression	Integer representation	Binary representation
---	3	0000000000000011
---	5	0000000000000101
3 EQV 5	-7	1111111111111001

## Functions

A function executes predefined operations on specified values (called arguments) and returns a numeric value or character string as the result. Although functions are handled as expressions, unlike numeric, character, relational, and logical expressions the function itself holds the result of the operations.

Some functions are automatically assigned values by the system. These functions are known as system variables.

System variables require no arguments. Values are assigned to some system variables under special system conditions, such as when an error or interrupt occurs, but some other system variables always hold values, such as the time or date.

OVL offers user-defined functions that the BASIC user can define as required. User-defined functions are handled inside the program in the same way as the system variables.

Elementary functions (such as the SIN function) match the precision of the argument. The function becomes double precision if the argument is a double-

precision value or single-precision if the argument is an integer or a single-precision value.

### 2-6-3 Priority of Operations

Operations are executed in the order of priority shown below. Low numbers take priority over higher numbers. Operations with the same number are executed in the order in which they appear.

- 1, 2, 3...
1. Expressions enclosed in parenthesis
  2. Functions
  3. ^ (Exponents)
  4. – (Minus signs: not preceded by a value or numeric expression)
  5. \*, / (multiplication and real number division)
  6. ¥ (integer division)
  7. MOD (remainder calculation)
  8. +, – (addition, subtraction)
  9. Relational operators (<, >, =, and combinations)
  10. NOT
  11. AND
  12. OR
  13. XOR
  14. IMP
  15. EQV

Example:

$$A\% = 2 + 8 \text{ MOD } 5 - 3$$

The above expression is interpreted as  $2 + (8 \text{ MOD } 5) - 3$ , so that  $A\% = 2$ . Although the parentheses are not strictly necessary in this example, it is normal to include them to prevent confusion when reading the program.

## 2-7 Interrupts

OVL supports the interrupts listed below.

Stop Key	(ON STOP GOSUB)
Help Key	(ON HELP GOSUB)
Real-time timer	(ON TIME\$ GOSUB)
Function Key	(ON KEY GOSUB)
RS-232C circuit	(ON COM GOSUB)
Processing error	(ON ERROR GOTO)
STEP signal	(ON INTR GOSUB)

## 2-8 Labels

Labels can be used instead of line numbers to control jump destinations in a program. Precede label names by an asterisk (\*).

Define the name after the asterisk in alphabetic characters and the period character (.). Uppercase and lowercase characters are not differentiated.

Reserved words cannot be used as label names. However, label names may contain reserved words.

The length label name is restricted only by the number of characters in the program line (255 characters).

Label names must be positioned at the start of the program line.

Example:

```
100 IF A<>B GOTO *UNEQUAL
110 ....
120 ....
200 *UNEQUAL
210 ....
```

\*UNEQUAL is interpreted in the GOTO statement as identical to 200. In this case, the program jumps to line 200 if  $A \neq B$ .



# SECTION 3

## Screen Operations

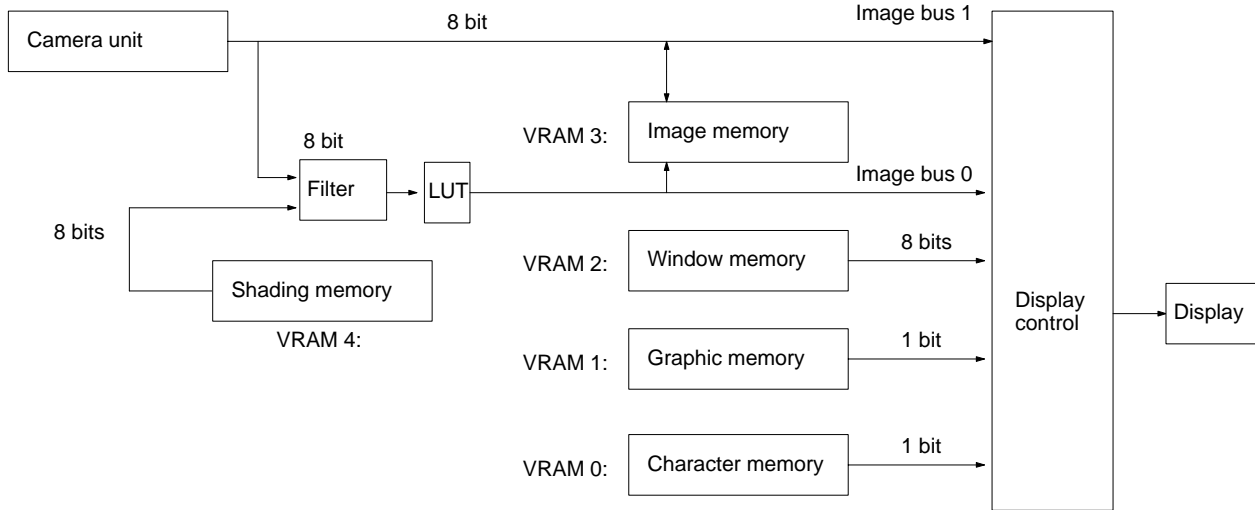
This section provides general information on screen manipulation.

3-1	Flow of Image Data .....	36
3-1-1	VRAM .....	36
3-1-2	LUT .....	38
3-1-3	Display LUT .....	39
3-1-4	Filter .....	39
3-2	Display and Measurement Block Diagrams .....	40
3-2-1	Display and Measurement Status of the Camera Image .....	40
3-2-2	Display and Measurement Status of the Image Memory Contents .....	41
3-3	Flow of Measured Data .....	41
3-4	Plane and Frame Memory .....	43
3-4-1	Plane Memory .....	43
3-4-2	Frame Memory .....	43
3-4-3	Binary Image Planes .....	43
3-4-4	Mask Bits .....	43
3-5	Windows .....	44
3-5-1	Window Planes .....	44
3-5-2	Paint Window .....	45
3-5-3	Pattern Matching Window .....	45
3-6	Drawing Density and Drawing Modes .....	46
3-6-1	Drawing Density .....	46
3-6-2	Drawing Mode .....	46
3-7	Run Length .....	47
3-7-1	Simple Run Length .....	47
3-7-2	Detailed Run Length .....	48
3-8	Labelling .....	48
3-9	Scrolling .....	49
3-9-1	Window Scrolling .....	49
3-9-2	Shading Scrolling .....	49
3-10	Fill Measurement .....	50
3-11	Image Cut-off .....	50
3-12	Scan Measurement .....	51

### 3-1 Flow of Image Data

OVL offers image-processing commands and functions in addition to the normal BASIC commands and functions. The flow of this image-processing data is shown in the block diagram below.

#### Image Data Flow



#### 3-1-1 VRAM

Memory directly influencing the measurements or display is known as VRAM. The F300 has 5 types of VRAM, which are explained below.

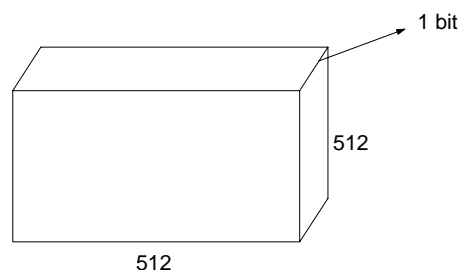
Memory	Type
0: Character memory	Plane
1: Graphic memory	
2: Window memory	Frame
3: Image memory	
4: Shading memory	

#### Character Memory (Plane)

The character memory is mainly used to display characters and is used to edit the OVL program and display compilation results. The contents of the character memory have no direct effect on the measurement results. The memory is configured as 512 x 512 x 1 bit.

Density (0 or 1 only):

0 A 0  
!0 A 1



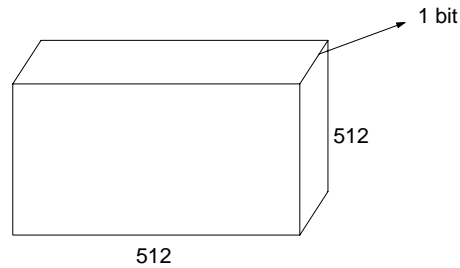
**Graphic Memory (Plane)**

The application of the graphic memory is not fixed.

Density (0 or 1 only):

0 A 0

!0 A 1



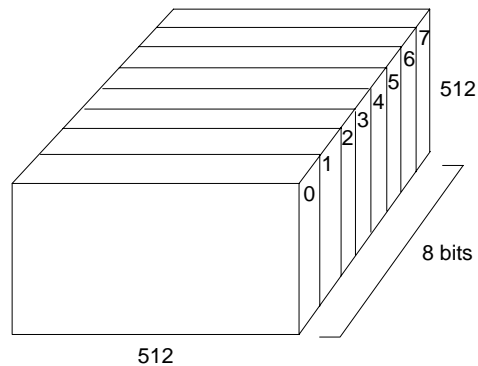
**Window Memory (Frame)**

The window memory is used to draw the windows. It consists of 8 window planes with a one-to-one relationship to the binary image planes.

The window memory can be treated as 8 memory areas of 512 x 512 x 1 bit configuration or as a single area with 512 x 512 x 8 bit configuration.

Density:

0 to 255



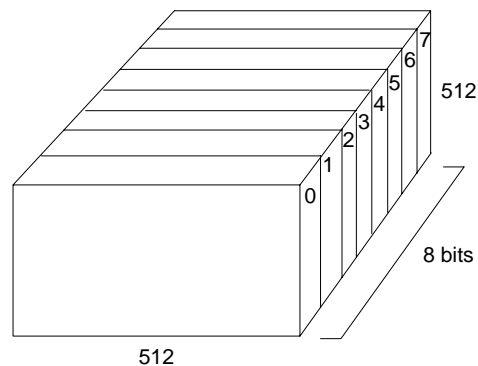
**Image Memory (Frame)**

The image memory is used to store the raw image direct from the camera or the 8 binary images generated from the look-up table (LUT). The image memory contents are treated as the raw measurement data.

The image memory can be treated as 8 memory areas of 512 x 512 x 1 bit configuration or as a single area with 512 x 512 x 8 bit configuration.

Density:

0 to 255

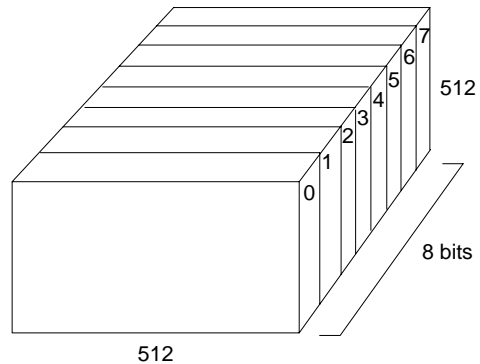


### Shading Memory (Frame)

The shading memory is used for shading compensation. If no shading compensation is carried out, the memory has no effect on the measurements or display.

The shading memory can be treated as 8 memory areas of 512 x 512 x 1 bit configuration or as a single area with 512 x 512 x 8 bit configuration. The shading master memory has two banks: Bank 0 and Bank 1.

Density:  
0 to 255



### 3-1-2 LUT

LUT is an abbreviation of Look-Up Table. The LUT is a conversion table for selecting the colors displayed by the display system. The LUT plays the same role in the system as a palette plays for an artist painting a picture; it selects the actual color from amongst numerous gradations.

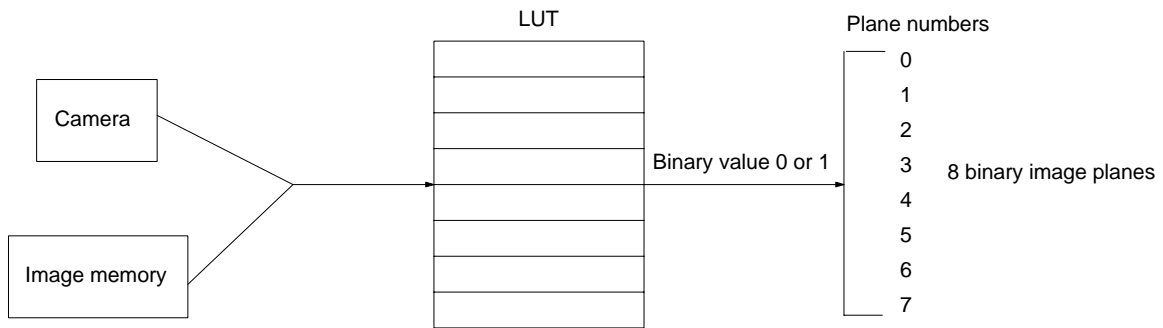
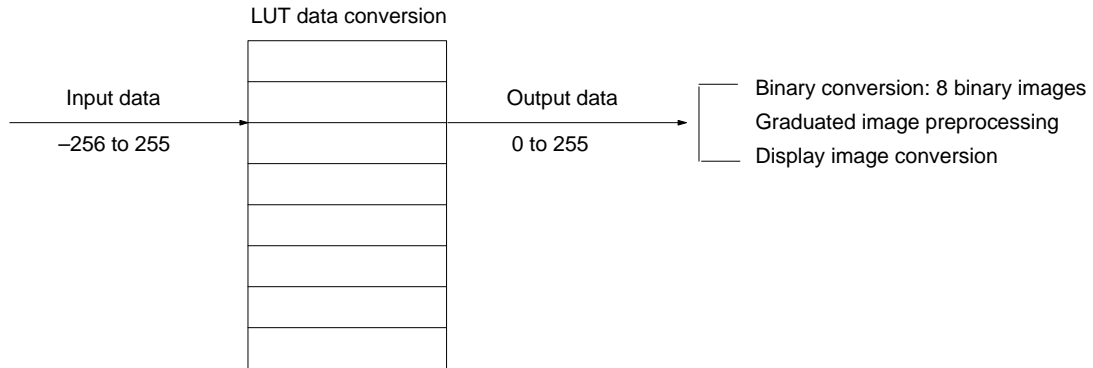
To express monochrome gradations, the OVL assigns 8 bits of data to each image pixel, that is  $2^8 = 256$  gradations.

The LUT is used with the frame-structured memories (VRAM3: image memory, VRAM4: shading memory). The gradation in the frame memory is determined by referring to the LUT.

The LUT is a prewritten memory containing output data corresponding to any input data address. The F300 uses the LUT conversion function for preprocessing of binary and raw images and for conversion of the displayed image.

Related commands and functions:

AUTOLVL	ENHANCE
GETBLUT	GETLUT
HISTGRAM	LEVEL
SETBLUT	SETLUT
SFTBLUT	SFTLUT



### 3-1-3 Display LUT

The LUT used to improve the contrast of the display image and change the gradations (0 to 255) of the raw image inside or outside a window is known as the display LUT.

Related commands:

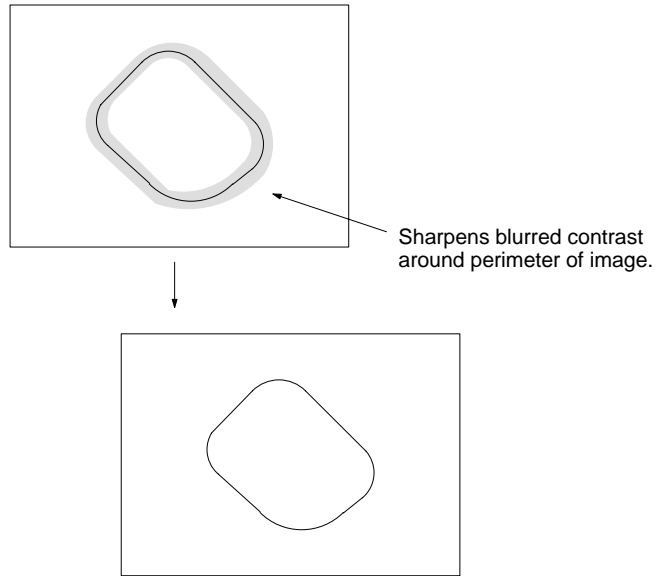
GETDLUT
SETDLUT

### 3-1-4 Filter

The filter is a pre-processing feature which sharpens the edges when the perimeter of the measured object is blurred.

Related commands:

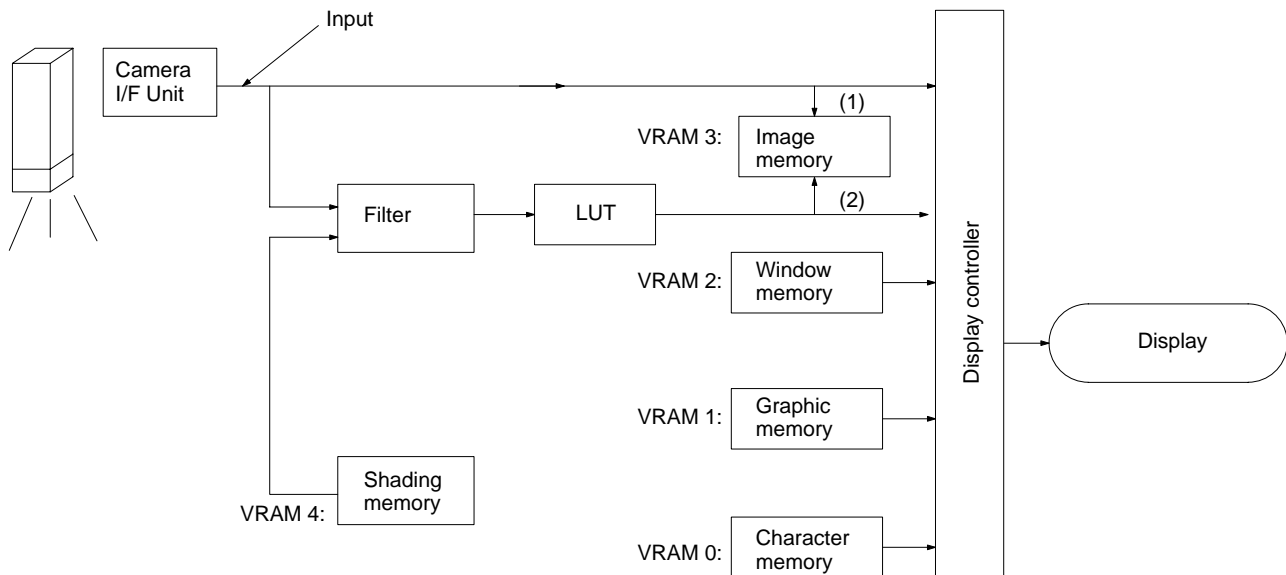
FILTER  
FILTDATA



## 3-2 Display and Measurement Block Diagrams

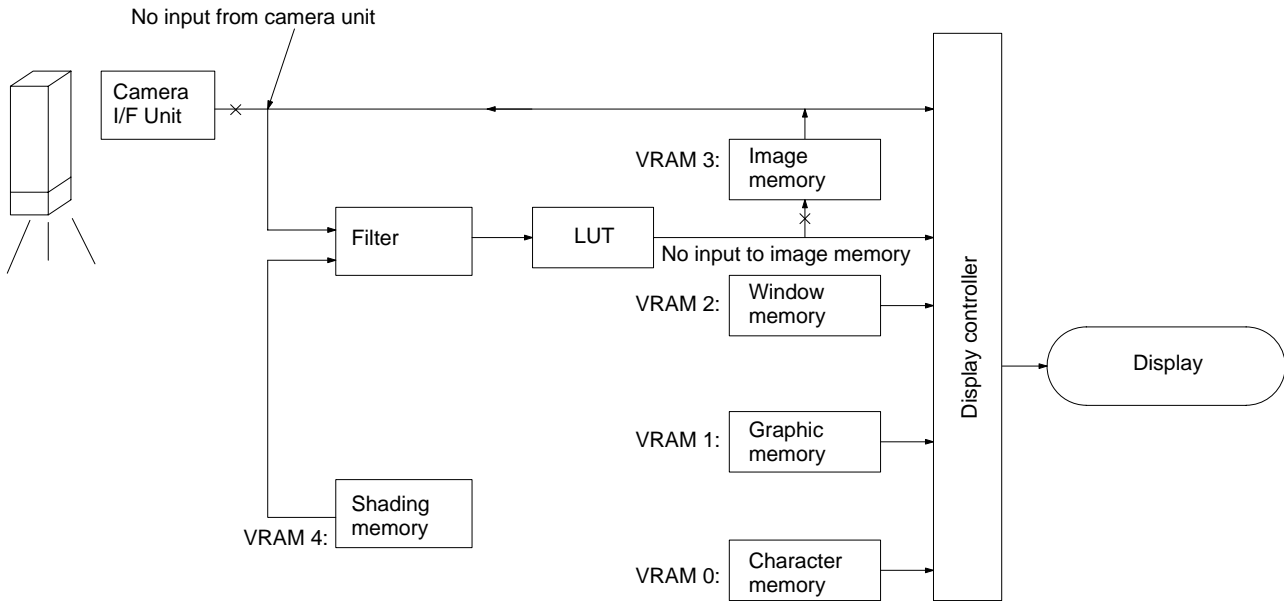
### 3-2-1 Display and Measurement Status of the Camera Image

The image data recorded by the camera is input to the image memory either directly from the camera ((1) in the diagram) or after LUT conversion of the image ((2) in the diagram).



### 3-2-2 Display and Measurement Status of the Image Memory Contents

Input from the camera unit is not possible during measurement or display of an image already stored in memory.



### 3-3 Flow of Measured Data

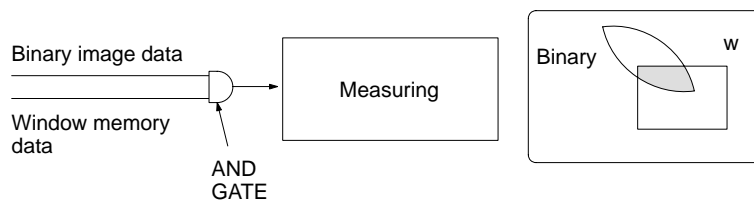
This section describes how the combination of the window function (W), paint function (P), and pattern matching function (PM) set with the measure mode (MMODE) command acts on the binary image planes 0 to 7.

- W: Window function
- P: Paint function
- PM: Pattern matching function

The functions themselves are described in detail in 3-5 Windows.

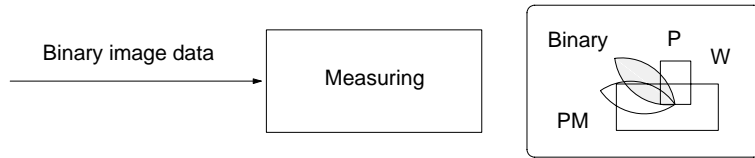
#### Setting Example 1

Function	Setting	Description
W	On	Window function: ON
P	Off	Paint function: OFF
PM	Off	Pattern matching function: OFF



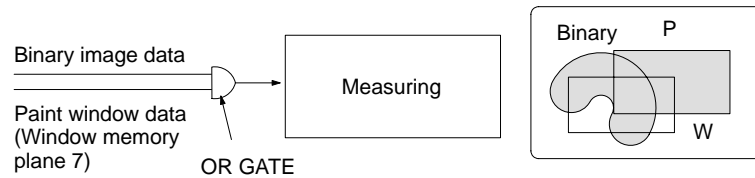
Setting Example 2

Function	Setting	Description
W	Off	Window function: OFF
P	Off	Paint function: OFF
PM	Off	Pattern matching function: OFF



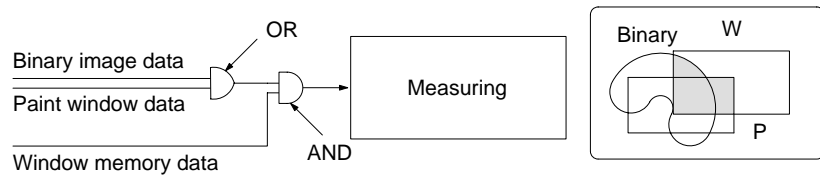
Setting Example 3

Function	Setting	Description
W	Off	Window function: OFF
P	On	Paint function: ON
PM	Off	Pattern matching function: OFF



Setting Example 4

Function	Setting	Description
W	On	Window function: ON
P	On	Paint function: ON
PM	Off	Pattern matching function: OFF



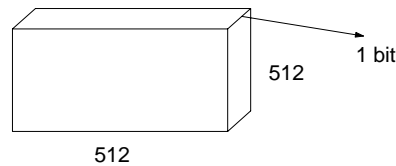


## 3-4 Plane and Frame Memory

### 3-4-1 Plane Memory

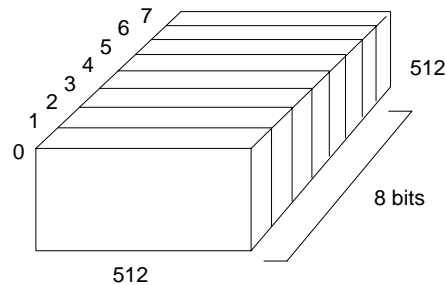
The term “plane” is used to describe a memory with 512 x 512 x 1 bit configuration. Consequently, the character memory, graphic memory, and parts of the other 3 image memories with 512 x 512 x 1 bit configuration are grouped under the name “plane memories.”

VRAM	Type
0: Character memory 1: Graphic memory	Plane (512 x 512 x 1 bit)
2: Window memory 3: Image memory 4: Shading memory	Frame (512 x 512 x 8 bit)



### 3-4-2 Frame Memory

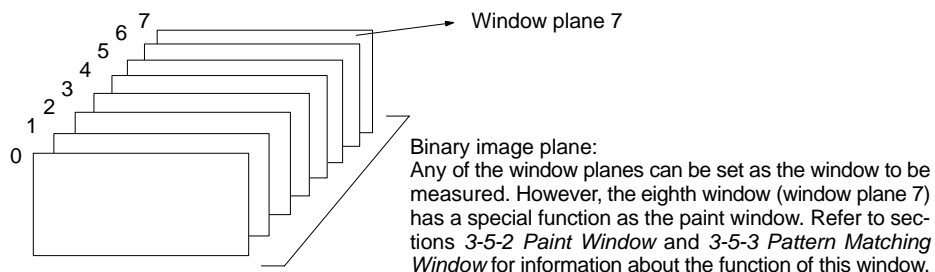
The term “frame” is used to describe an image memory with 512 x 512 x 8 bit configuration. Consequently, the window memory, image memory, and shading memory with 512 x 512 x 8 bit configuration memories are grouped under the name “frame memories.”



### 3-4-3 Binary Image Planes

Each of the eight types of binary image converted by the binary LUT from the camera or frame memory raw image is known as a binary image plane, and is denoted by a binary plane number between 0 and 7. The binary image plane has a one-to-one relationship with the window plane of the same number. All eight binary image planes are measured simultaneously.

Often the term “plane” is used to include the window plane.



### 3-4-4 Mask Bits

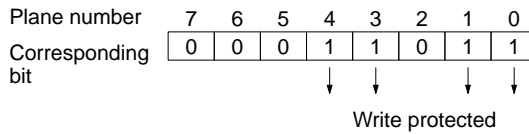
Write protection can be enabled by setting a mask bit to prevent data being changed after it is written to a frame memory.

Related commands:

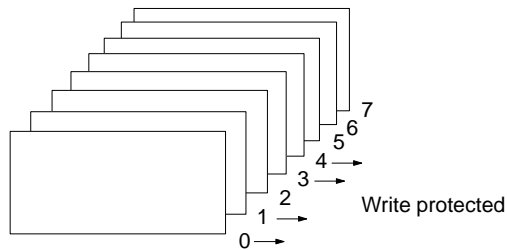
MASKBIT

VRAM	Type	Description
0: Character memory 1: Graphic memory	Plane	Mask bits not available
2: Window memory 3: Image memory 4: Shading memory	Frame	Mask bit can be used to enable write protection.

Example:  $27_{(10)} = 00011011_{(2)}$



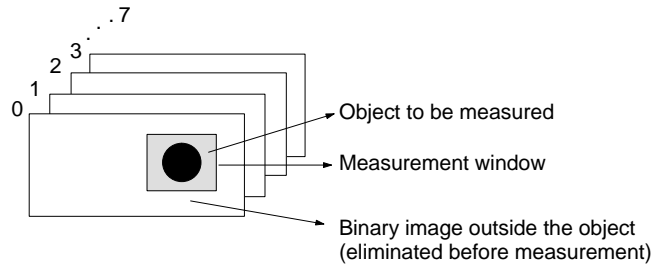
**Note** If the mask bit is set to 1, the corresponding plane is write protected and its contents cannot be changed.



In the example above, planes 0, 1, 3, and 4 are write protected. The other planes are write enabled, that is the data can be changed.

## 3-5 Windows

Windows define the measurement area on a binary image plane. If some of the binary image displayed on the screen is not to be measured, a window graphic can be set to include only the object to be measured so that the binary image outside the object is deleted before measurement.

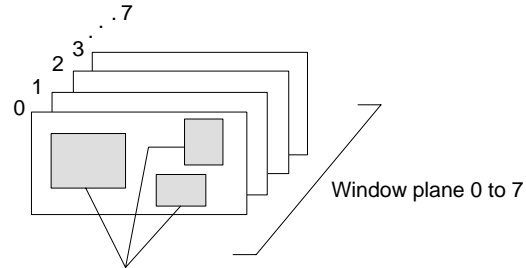


### 3-5-1 Window Planes

Multiple windows can be set for each of the eight binary image planes. In this case, each of the eight binary image planes is known as a window plane and has the same number between 0 and 7 as the binary image plane.

The term “plane” may be used to refer collectively to binary image planes and window planes.

The eight binary image planes:



Multiple windows can be set for each binary image plane. In this example, three windows are set in a single window plane.

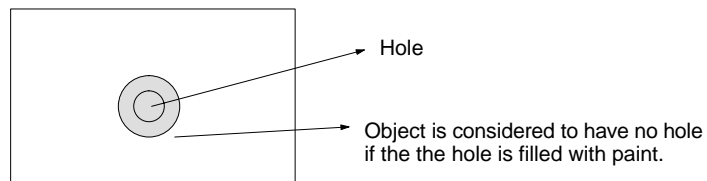
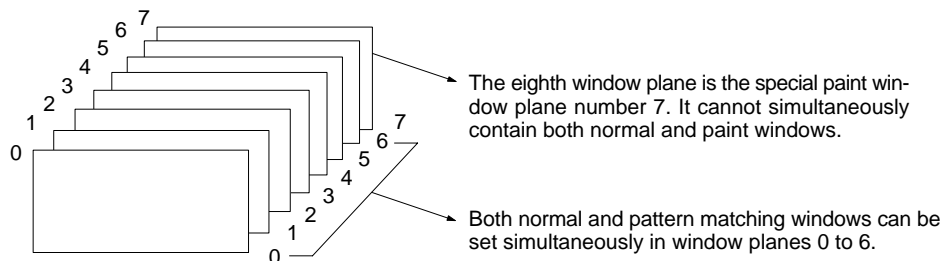
### 3-5-2 Paint Window

The eighth window (window plane 7) is used exclusively for paint windows. A paint window can be set in the area to be measured, regardless of whether the binary image of the measured object is available. For example, by selecting a paint window over the hole in the measured object, the object is measured as if it had no hole.

Window plane 7 can be set as the paint window for each binary image plane. Each of the binary image planes number 0 to 6 can function simultaneously with normal windows and paint windows. However, normal and paint windows cannot be mixed in binary plane number 7.

A paint and pattern matching window cannot be set simultaneously in a single binary image plane.

The eight binary image planes:



### 3-5-3 Pattern Matching Window

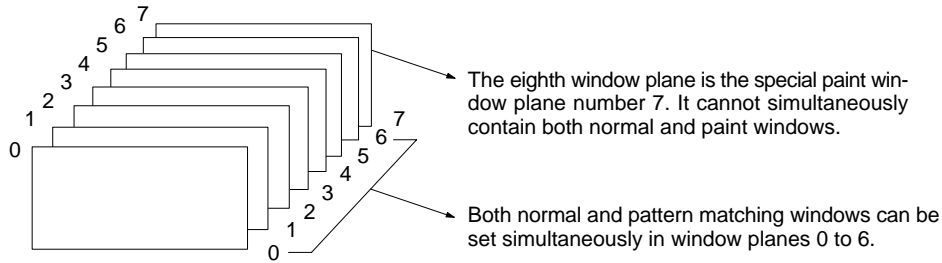
Pattern matching is carried out in window plane 7.

In this case, window plane number 7 is known as the pattern matching window. During pattern matching, the shape of the pattern matching window is superimposed on the shape of the measured object binary image and the area of the non-matching parts is measured.

Window plane 7 can be set as the pattern matching window for each binary image plane. Each of the binary image planes number 0 to 6 can function simultaneously with normal windows and pattern matching windows. However, normal and paint windows cannot be mixed in binary plane number 7.

A paint and pattern matching window cannot be set simultaneously in a single binary image plane.

The eight binary image planes:



### 3-6 Drawing Density and Drawing Modes

#### 3-6-1 Drawing Density

The drawing density directly specifies the density when drawing a graphic to image memory with the graphic control command. Existing contents of the image memory are overwritten.

Only values 0 and 1 are written to a plane image memory. In this case, specify the drawing density either as 0 or as a non-zero value (≠0). The value 0 is written to memory if the drawing density is specified as 0. The value 1 is written to the image memory if the drawing density is specified as a non-zero value.

Drawing density values between 0 and 255 can be written to a frame image memory. These values are expressed as 8-bit binary values. Each of these bits corresponds to one plane of the frame memory. For example, if the drawing density is 105, which is expressed as 01101001 in binary, 1 is written to planes 0, 3, 5, and 7, and 0 is written to planes 1, 2, 4, and 6.

VRAM	Type	Density
0: Character memory 1: Graphic memory	Plane	0 or 1
2: Window memory 3: Image memory 4: Shading memory	Frame	0 to 255

Example:  $105_{(10)} = 01101001_{(2)}$

Plane number	7	6	5	4	3	2	1	0
Corresponding bit	0	0	0	1	1	0	1	1

0: Off  
1: On

#### 3-6-2 Drawing Mode

The drawing mode carries out logical operations on the graphic in the graphic memory and the graphic data being written to the graphic memory when a graphic is drawn to image memory with the graphic control command.

The following three modes can be specified:

- OR: the drawn graphic is ORed with the image memory contents
- XOR: the drawn graphic is XORed with the image memory contents
- NOT: the contents of the image memory are cleared

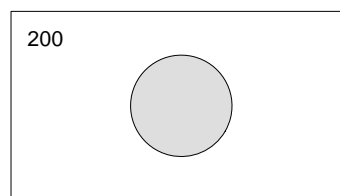
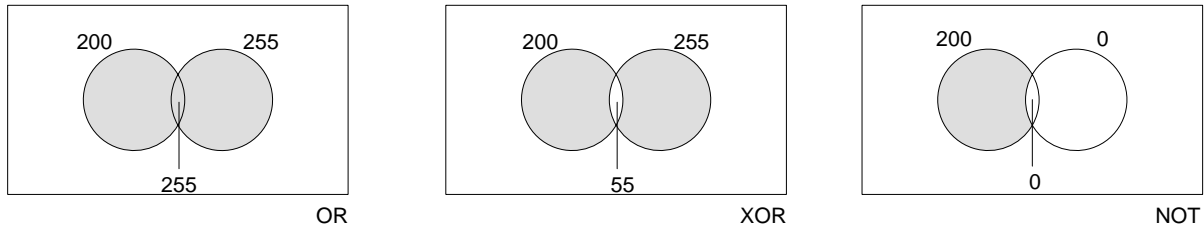


Image memory contents = 200

The results of these logical operations are shown below.

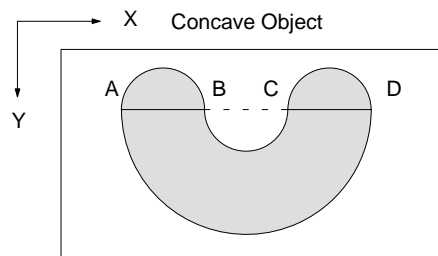


### 3-7 Run Length

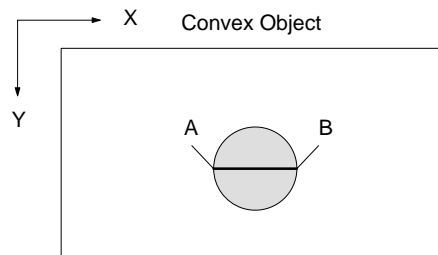
A binary image can be considered as a collection of line segments in the X and Y directions. The length of these line segments is known as the run length. The F300 can measure two types of run length: the simple run length and the detailed run length.

Detailed run length: The lengths of lines AB and CD are determined separately.

Simple run length: The total length of lines AB and CD is determined.



Both the simple run length and detailed run length give the same result for the length of the line segment AB.



#### 3-7-1 Simple Run Length

The simple run length is the total length of line segments at various Y-coordinate positions. Measurement is possible in the X direction only.

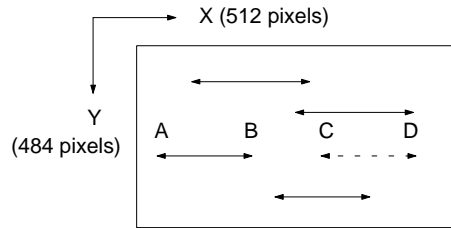
If multiple line segments exist at a single Y coordinate, the individual lengths of each section cannot be determined. If measurements are made on a convex object with no hole, the simple run length is identical to the detailed run length in the X direction.

If a concave object is measured, the simple run length includes the detailed run length.

Related commands and functions:

- MEASURE
- MMODE
- RUNL

It is not possible to determine the individual lengths of more than one line segment at a single Y-axis coordinate axis position, like AB and CD in the diagram for example.

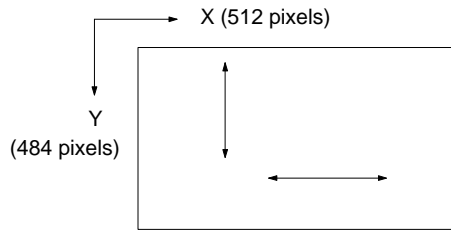


### 3-7-2 Detailed Run Length

The detailed run length is the individual length of each line segment at an X- or Y-coordinate position. The number of detailed run length data depends on the size of the binary image and complexity of the shape.

Related commands and functions:

- MEASURE
- RMODE
- RDATA

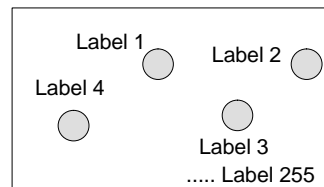


## 3-8 Labelling

Labelling involves applying serial numbers starting from 0 to unlinked binary images. Complex applications can be handled using the results of labelling and other measurement properties.

Labelling can be used to determine how many measured objects are contained in the screen. The maximum label number corresponds to the number of objects. The properties of each object can be determined from the data after labelling, giving more detailed information.

- Label 1: Measured object 1
- Label 2: Measured object 2
- Label 3: Measured object 3
- Label 4: Measured object 4
- Label 255: Measured object 255

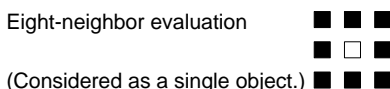
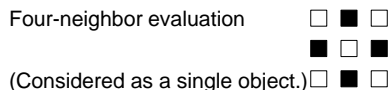


Two methods are used to determine the linked status of objects during labelling: Four-neighbor evaluation and eight-neighbor evaluation.

In four-neighbor evaluation, pixels are considered to be linked into a single object if the four pixels on the top, bottom, left, and right of the evaluated pixel

have the same density. The pixels in the diagonal corners of the evaluated pixel are ignored.

In eight-neighbor evaluation, pixels are considered to be linked into a single object if all adjacent pixels have the same density.

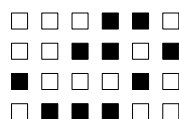


■ : These pixels all have the same density.

Example:

Four-neighbor evaluation: Considered as five objects

Eight-neighbor evaluation: Considered as one object



Related commands:

- LABEL            LPUTIMG
- LSORT            LDATA
- LPOINT

## 3-9 Scrolling

### 3-9-1 Window Scrolling

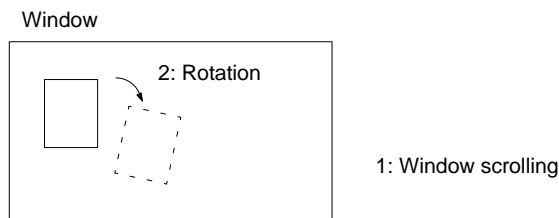
If the position of the measured object moves inside the screen, the position of the window must be moved accordingly. This operation is known as positional displacement compensation.

The following two methods are possible for compensating for positional displacement:

- Compensation by scrolling the window
- Rotating the image of the measured object

The F300 uses only the first method, compensation by scrolling the window, as this method overcomes the errors in measurement results which occur when the image of the measured object is rotated.

Related command:    WSCROLL

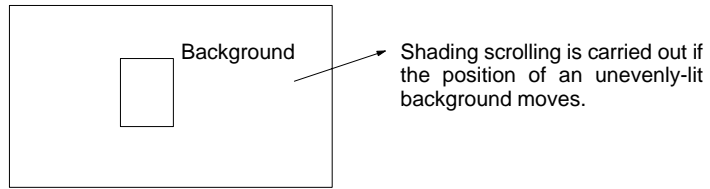


### 3-9-2 Shading Scrolling

If the background to the measured object has uneven brightness and the position of the background changes inside the screen, shading compensation is required to move the shading memory contents accordingly. This operation is known as shading scrolling.

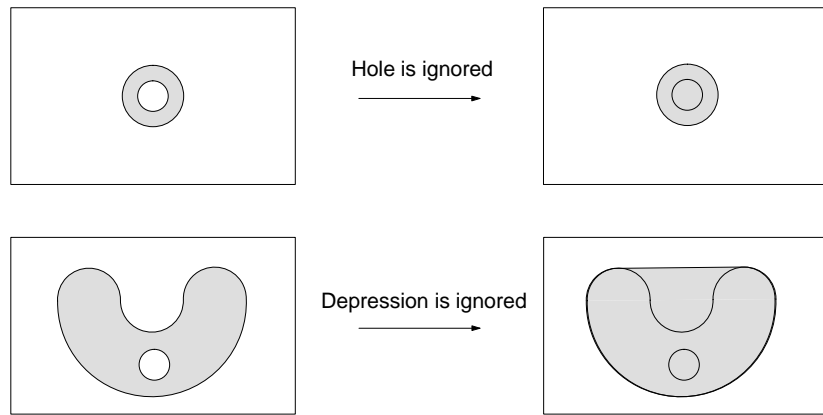
Related command:

SSCROLL



### 3-10 Fill Measurement

The measurement of an object containing a hole as if it had no hole is known as fill measurement. However, accurate measurements ignoring the hole only are only possible for measured objects with a convex outline. A concave outline is defined as a shape in which no tangent crosses the outline itself. Consequently, a hole in a circle, triangle, or square outline is ignored accurately, but this is not possible with a star-shaped or U outline, for example.

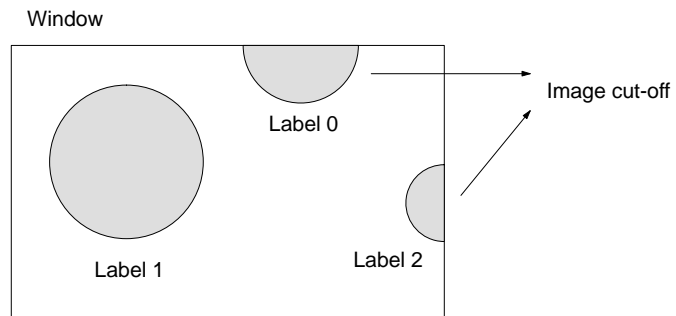


### 3-11 Image Cut-off

Removing images which coincide with the edge of the window is known as image cut-off. Images fully contained in the window remain unchanged. The F300 uses labelling to carry out this feature.

Related commands:

RMODE LABEL  
 LPUTIMG LSORT  
 LDATA LPOINT



During labelling, cut-off images are skipped, so that in this case only Label 1 remains.



## 3-12 Scan Measurement

Scan measurement allows measurement of the number and length of measured objects along the binary image of any shaped line in the image memory.

The following shapes can be scanned:

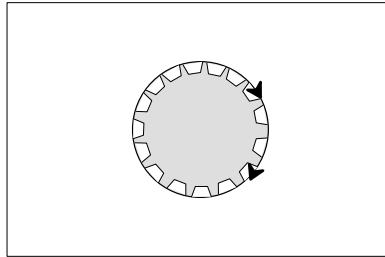
- Ellipses (including circles)
- Polygons
- Straight lines
- Polygonal lines

Related commands and functions:

SCAN	SCANSET
SDATA1	SDATA2

Example:

The number of teeth and various scan lengths can be measured on this gearwheel.



## **SECTION 4**

### **Reference**

This section provides detailed information on the commands and functions. Examples are also provided.

**ABS****ABSolute**

(Function)

<b>Action</b>	Determines an absolute value.
<b>Format</b>	ABS ( <i>numeric expression</i> )
<b>Example</b>	A = ABS (-7) Assigns 7, the absolute value of -7, to variable A.
<b>Description</b>	Determines the absolute value of the specified <i>numeric expression</i> . The <i>numeric expression</i> may be in the form of an integer, long integer, or a single or double-precision real number. The ABS function always returns a double-precision real number.

**AKCNV\$****Ank Kanji CoNVert\$**

(Function)

<b>Action</b>	Converts 1-byte characters in a character string to 2-byte characters.
<b>Format</b>	AKCNV\$ ( <i>character string</i> )
<b>Example</b>	C\$ = AKCNV\$ ("OVL") Assigns "OVL" to variable C\$.
<b>Description</b>	The AKCNV\$ function converts all 1-byte characters in a <i>character string</i> to a 2-byte equivalent. The AKCNV\$ function returns a character string containing 2-byte characters only. Specify a <i>character string</i> containing a mixture of 1- and 2-byte characters as a character variable or character constant with the character string variable. The KACNV\$ function has the opposite action to the AKCNV\$ function. KACNV\$ converts 2-byte characters in a <i>character string</i> to a 1-byte equivalent.

**ARC****ARC**

(Command)

<b>Action</b>	Draws an arc in VRAM.
<b>Format</b>	ARC X, Y, R, <i>start angle</i> , <i>end angle</i> , VRAM [, [page#] [, <i>drawing density or drawing mode</i> ]]
<b>Example</b>	ARC 200, 250, 45, 35, 75, 2, , 128 Draws an arc on Plane 7 of the window memory with center coordinates (200, 250), radius (45), start angle (35%), and end angle (75%).
<b>Description</b>	The ARC command draws an arc clockwise between the <i>start</i> and <i>end angle</i> around the <i>center coordinates</i> (X, Y) with the specified <i>radius</i> (R). <i>Start angle</i> and <i>end angle</i> specified in degrees. Specify the VRAM where the arc is drawn with a number, as follows. 0: Character memory 1: Graphic memory 2: Window memory 3: Image memory 4: Shading memory Omit the <i>page#</i> or set to 0. Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i> . The default value is <i>drawing mode</i> , OR.

The *drawing density* parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The *drawing density* parameter has the following effect when set for the character or graphic memory.

0 : 0 written to memory

!0: 1 written to memory

The *drawing mode* settings operate as follows.

OR: The current contents of the image memory ORed with 255 are written to memory.

NOT: 0 is written to memory

XOR: The current contents of the image memory are inverted.

The default value for the drawing mode is OR.

When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

## ASC

## ASCIi

(Function)

<b>Action</b>	Determines the character code of the first character in a character string.
<b>Format</b>	ASC ( <i>character string</i> )
<b>Example</b>	A = ASC (" BASIC") Assigns the character code (66) for the first letter (B) of the character string (BASIC) to variable A.
<b>Description</b>	Returns the decimal character code of the first character in the specified <i>character string</i> . The CHR\$ function has the opposite action to the ASC function. CHR\$ returns the character corresponding to the specified character code.

## ATN

## ArcTanGent

(Function)

<b>Action</b>	Determines the arctangent (arctan) of a value.
<b>Format</b>	ATN ( <i>numeric expression</i> )
<b>Example</b>	A = ATN(1)*180/π Determines the arctan of 1 and assigns the corresponding angle (45) to variable A.
<b>Description</b>	The ATN function returns the arctangent value in radians between $-p/2$ and $p/2$ . Multiply this value by $180/p$ to convert the returned value to degrees. The <i>numeric expression</i> may be in the form of an integer, long integer, or a single or double-precision real number. The ATN function always returns a double-precision real number.

## ATTR

## ATTRibute\$

(Function)

<b>Action</b>	Determines the write protect attribute of the specified file.
<b>Format</b>	ATTR.\$ ( <i>filename or [#] file#</i> )
<b>Example</b>	ATTRIBUTE\$ = ATTR\$(" FILE.BAS") Determines the write protect attribute of the file FILE.BAS and assigns the result to variable ATTRIBUTE\$.

**Description** Determines if the file specified with a *filename* or *file#* is write protected or write enabled and returns a letter indicating the status.

Specify the *filename* stored in a memory card as a character string. Specify the *file#* as the number in which the file was opened with the OPEN statement.

The ATTR\$ function returns a 2-character character string (attribute code) corresponding to the write protect attribute assigned to the file specified with the *filename* or *file#*. The attribute codes are described in the table below.

Attribute code	Meaning
"_"	The file has no write protect attribute allocated. The file specified with the <i>filename</i> or <i>file#</i> is write enabled.
"_P"	The file specified with the <i>filename</i> or <i>file#</i> is write protected.
"E_"	The file specified with the <i>filename</i> or <i>file#</i> is coded.

**Note** \_ indicates a blank

## AUTO

## AUTOMATIC numbering

(Command)

**Action** Automatically generates line numbers when the program is keyed in.

**Format** AUTO [*initial value*] [,*increment*]

**Example** AUTO 1000, 20  
The line numbers 1000,1020, 1040, ... are displayed automatically.

**Description** When the AUTO command is input, the line number *initial value* is displayed at the start of the next line and the cursor stops to the right of this line number. Subsequently, each time the Return Key is pressed to input a line, the present line number plus the specified *increment* is displayed at the start of the next line.

The default value of the *initial value* is 10. The default value of the *increment* is 10. Both the *initial value* and *increment* may be omitted.

If a line number already exists, an asterisk (\*) is displayed after the line number. Press the CTRL+C or STOP Keys to end automatic line number generation.

## AUTOLVL

## AUTO LeVeL

(Function)

**Action** Determines the ideal binary level from density histogram data.

**Format** AUTOLVL (*algorithm*, *array name*, *qualifier*)

**Example** AL = AUTOLVL (0, H, 0)  
The ideal binary level will be obtained from the histogram data of array H.

**Description** The AUTOLVL determines the ideal binary level from the array data, which is specified with the *array name*, where the histogram data obtained with the HISTGRAM command is stored.

Always set the *algorithm* to 0.

The *qualifier* specifies the first element used in the histogram data array. Normally set to 0.

## BACKDISP

## BACKground DISPLAY

(Command)

**Action** Sets the type of display for the image outside the window.

**Format** BACKDISP *display image* [, [*binary image plane#*] [, *binary reverse*]]

**Example** BACKDISP 2  
Sets a mask image (black) for the image outside the window.

<b>Description</b>	<p>BACKDISP sets how the image outside the window is displayed.</p> <p>Set the <i>display image</i> to one of the following values.</p> <ul style="list-style-type: none"> <li>0: Raw image</li> <li>1: Binary image</li> <li>2: Mask image (black)</li> </ul> <p>If a binary image is specified as the <i>display image</i>, set the <i>binary image plane number</i> between 0 and 7 to specify the plane for binary conversion. The binary image corresponding to the specified binary image plane# is displayed outside the window. The default value is 0.</p> <p>The binary reverse value can be specified if a binary image is selected, as follows.</p> <ul style="list-style-type: none"> <li>0: Normal monochrome display</li> <li>1: Reversed monochrome display</li> </ul> <p>The default value is 0.</p> <p>The BACKDISP command sets the display only and does not affect the measurements.</p>
--------------------	---

**BCOPY****Binary COPY**

(Command)

<b>Action</b>	Copies a binary image in VRAM.
<b>Format</b>	BCOPY VRAM1, [page# 1], [plane# 1], VRAM2, [page# 2], [plane# 2] [, [X1] [, [Y1] [, [X2] [, [Y2] [, [X] [, Y]]]]]
<b>Example</b>	<p>BCOPY 2,, 0, 2,, 1</p> <p>Copies the entire Plane 0 in window memory to Plane 1.</p>
<b>Description</b>	<p>BCOPY copies a binary image between VRAMs.</p> <p>Specify the numbers of the copy source and destination VRAMs, with <i>VRAM1</i> and <i>VRAM2</i>, as follows.</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the setting for the <i>page# 1</i> and <i>page# 2</i> or set them to 0.</p> <p>Specify the plane number of the copy source and destination VRAMs, with <i>plane# 1</i> and <i>plane# 2</i>.</p> <p>To enclose the copy source image in a rectangular region, specify the top-left and bottom-right coordinates as <i>X1</i>, <i>Y1</i> and <i>X2</i>, <i>Y2</i>, respectively. The default value of <i>X1</i> and <i>Y1</i> is 0 and the default value of <i>X2</i> and <i>Y2</i> is 511.</p> <p>Specify the top-left coordinates of the copy destination copy start region with <i>X</i>, <i>Y</i>. The default values of <i>X</i>, <i>Y</i> are 0,0.</p> <p>When copying from a plane VRAM to a frame VRAM, if the <i>plane#</i> of the copy destination (<i>plane# 2</i>) is specified, only the specified plane is copied, otherwise all planes are copied.</p> <p>The <i>plane#</i> of the copy source VRAM (<i>plane# 1</i>) must be specified when copying from a frame VRAM to a plane VRAM. Data in planes write protected with the MASKBIT command remains unchanged when data is copied to a frame VRAM.</p>

**BCOPY2****Binary COPY2**

(Command)

<b>Action</b>	Makes enlarged and reduced copies of binary images in VRAM.
<b>Format</b>	<code>BCOPY2 VRAM1, [page# 1], [plane# 1], VRAM2, [page# 2], [plane# 2] [, [XS1] [, [YS1] [, [XS2] [, [YS2] [, [XD1] [, [YD1] [, [XD2] [, YD2]]]]]]]]]]</code>
<b>Example</b>	<p><code>BCOPY2 2,, 0, 2,, 1, 100, 100, 200, 200, 200, 200, 400, 400</code></p> <p>Enlarges by a factor of 2 the rectangular region with corner coordinates (100,100), (200,200) on plane 0 of the window memory and copies it to the rectangular region with corner coordinates (200,200), (400,400) on plane 1.</p>
<b>Description</b>	<p>BCOPY2 makes enlarged and reduced copies of binary images between rectangular regions in VRAM.</p> <p>Specify the numbers of the copy source and destination VRAMs, with <i>VRAM1</i> and <i>VRAM2</i>, as follows.</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the setting for the <i>page# 1</i> and <i>page# 2</i> or set them to 0.</p> <p>Specify the plane number of the copy source and destination VRAMs, with <i>plane# 1</i> and <i>plane# 2</i>.</p> <p>Specify the top-left and bottom-right coordinates of the copy source rectangular region as <i>XS1</i>, <i>YS1</i> and <i>XS2</i>, <i>YS2</i>, respectively. The default value of <i>XS1</i> and <i>YS1</i> is 0 and the default value of <i>XS2</i> and <i>YS2</i> is 511.</p> <p>Specify the top-left and bottom-right coordinates of the copy destination rectangular region as <i>XD1</i>, <i>YD1</i> and <i>XD2</i>, <i>YD2</i>, respectively. The default value of <i>XD1</i> and <i>YD1</i> is 0 and the default value of <i>XD2</i> and <i>YD2</i> is 511.</p> <p>When copying from a plane VRAM to a frame VRAM, if the plane# of the copy destination (<i>plane# 2</i>) is specified, only the specified plane is copied, otherwise all planes are copied.</p> <p>The plane# of the copy source VRAM (<i>plane# 1</i>) must be specified when copying from a frame VRAM to a plane VRAM. Data in planes write protected with the MASKBIT command remains unchanged when data is copied to a frame VRAM.</p>

**BEEP****BEEP**

(Command)

<b>Action</b>	Turns the buzzer on and off.
<b>Format</b>	<code>BEEP [switch]</code>
<b>Example</b>	<p><code>BEEP 1</code></p> <p>The buzzer sounds.</p>
<b>Description</b>	<p>The BEEP command turns the buzzer on and off.</p> <p>Specify the <i>switch</i> parameter as 0 or 1, as follows.</p> <ul style="list-style-type: none"> <li>0: Stop the buzzer.</li> <li>1: Continuously sound the buzzer.</li> </ul> <p>The buzzer sounds for a fixed time if the parameter is omitted.</p> <p>After the buzzer is turned on, it can be turned off again only with the BEEP command. The buzzer remains on if program operation stops.</p>

**BOX**

**BOX**

(Command)

<b>Action</b>	Draws a rectangle in VRAM.
<b>Format</b>	BOX X1, Y1, X2, Y2, VRAM [, [page#] [, [drawing density or drawing mode] [, lineart]]]
<b>Example</b>	BOX 0, 0, 128, 255, 2,, 128, 0 Draws a rectangle on Plane 7 of the window memory with opposing corner coordinates (0,0), (128,255).
<b>Description</b>	<p>The BOX command draws a rectangle between the opposing corner coordinates (X1, Y1) and (X2, Y2).</p> <p>Specify the VRAM where the rectangle is drawn with a number, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i>. The default value is <i>drawing mode</i>, OR.</p> <p>The <i>drawing density</i> parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The <i>drawing density</i> parameter has the following effect when set for the character or graphic memory:</p> <ul style="list-style-type: none"> <li>0 : 0 written to memory</li> <li>!0: 1 written to memory</li> </ul> <p>The <i>drawing mode</i> settings operate as follows.</p> <ul style="list-style-type: none"> <li>OR: The current contents of the image memory ORed with 255 are written to memory.</li> <li>NOT: 0 is written to memory</li> <li>XOR: The current contents of the image memory are inverted.</li> </ul> <p>The default value for the drawing mode is OR.</p> <p>Specify with the <i>lineart</i> parameter if the rectangle is an outline only or filled.</p> <ul style="list-style-type: none"> <li>0: Filled rectangle</li> <li>1: Rectangle outline only</li> </ul> <p>The default value is 0 (filled).</p> <p>When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.</p>

**BUSY**

**BUSY**

(Command)

<b>Action</b>	Controls the BUSY signal output status.
<b>Format</b>	BUSY 0 or 1
<b>Example</b>	BUSY 1 Turns ON the BUSY signal output of all connected Terminal Block Unit or Parallel I/O Units.
<b>Description</b>	<p>The BUSY command turns the BUSY signal output ON and OFF for the Terminal Block and Parallel I/O Units.</p> <p>Set the parameter to 0 to turn OFF the BUSY signal or to 1 to turn ON the BUSY signal.</p> <p>When multiple Terminal Block or Parallel I/O Units are connected, the BUSY signals are controlled simultaneously for all Units.</p>



**CALL****CALL**

(Command)

<b>Action</b>	Calls a subroutine defined between the SUB and END SUB commands.
<b>Format</b>	CALL <i>label</i> ( <i>argument</i> [, <i>argument</i> ..])
<b>Example</b>	CALL *SUB1 (PARAM1, PARAM2\$, 1) Calls the subroutine SUB1.
<b>Description</b>	The CALL command calls subroutines in which local variables are used. The <i>label</i> parameter specifies a <i>label</i> designated with the SUB command. The <i>arguments</i> specify data to be transferred to the subroutine. The <i>arguments</i> can be any type of variable, constant, or expression, but the argument type must match the type of <i>argument</i> used with the SUB command. If a variable specified as an <i>argument</i> is changed in the subroutine, the value of the <i>argument</i> changes simultaneously. No logical limitation is placed on the number of <i>arguments</i> . However, the command line can physically accommodate up to 255 characters only. An entire array cannot be used as an <i>argument</i> . Data from an array can be used as an <i>argument</i> by specifying individual elements of the array in the form: array name(qualifier). If multiple <i>arguments</i> are specified with the CALL command, no variable name can be used more than once.

**CAMCHK****CAMera CHECK**

(Function)

<b>Action</b>	Determines the connection status of the camera specified with the camera number.
<b>Format</b>	CAMCHK ( <i>camera</i> #)
<b>Example</b>	A = CAMCHK (1) Assigns the connection status of camera#1 to variable A.
<b>Description</b>	The CAMCHK function determines the status of the camera specified with the <i>camera number</i> (0 to 7), and returns a value as follows: 0: Connected -1: Not connected

**CAMERA****CAMERA**

(Command)

<b>Action</b>	Switches cameras.
<b>Format</b>	CAMERA <i>camera</i> #
<b>Example</b>	CAMERA 2 Switches to camera# 2.
<b>Description</b>	The CAMERA commands switches to the camera specified by the <i>camera number</i> (0 to 7). The F300 cannot simultaneously receive inputs from multiple cameras. Only the image from the camera selected with the CAMERA command is processed.

**CAMMODE****CAMera MODE**

(Command)

<b>Action</b>	Selects the image output from a Camera I/F Unit.
<b>Format</b>	CAMMODE <i>output image</i>
<b>Example</b>	CAMMODE 1 Selects the image from the Camera I/F Unit internal memory as the output image.

<b>Description</b>	<p>The CAMMODE command is valid for the Camera I/F Unit (normal/simultaneously) and Camera I/F Unit (shutter/simultaneously) only.</p> <p>Both the Camera I/F Unit (normal/simultaneously) and Camera I/F Unit (shutter/simultaneously) have a built-in image memory able to hold one image screen from a single camera. The CAMMODE selects if the image output from the Camera I/F Unit to the IMP Unit is the image output directly from the camera or the image from the Camera I/F Unit internal video memory.</p> <p>The parameter of the CAMMODE command selects the <i>output image</i>, as follows:</p> <ul style="list-style-type: none"> <li>0: Output the camera image directly.</li> <li>1: Output the image from the internal video memory.</li> </ul> <p>Images are written to the video memory with the FLASH command.</p> <p>If multiple Camera Units with internal memory are connected, the setting with the CAMMODE command applies to all Units.</p>
--------------------	---

**CDBL****Convert Double**

(Function)

<b>Action</b>	Converts a value to a double-precision value.
<b>Format</b>	CDBL ( <i>numeric expression</i> )
<b>Example</b>	<p>DBL# = CDBL(A!)</p> <p>Converts the value of the single-precision real number A! to a double-precision real number and assigns it to the double-precision real number variable DBL#.</p>
<b>Description</b>	<p>The CDBL function returns the specified numeric expression converted to a double-precision real number.</p> <p>The CDBL function converts values to a double-precision real value. It does not change the number of significant digits of the returned value. Consequently, the precision of the returned value is the same as the precision of the value before conversion.</p> <p>The following type conversion functions are also available:</p> <ul style="list-style-type: none"> <li>CINT function: Converts a value to an integer.</li> <li>CLNG function: Converts a value to a long integer</li> <li>CSNG function: Converts a value to a single-precision real number.</li> </ul>

**CHAIN****CHAIN**

(Command)

<b>Action</b>	Calls and transfers control to a specified program.
<b>Format</b>	<ol style="list-style-type: none"> <li>1. CHAIN "<i>filename</i>"[, <i>line#</i>] [, ALL]</li> <li>2. CHAIN MERGE "<i>filename</i>"[, <i>line#</i>] [, ALL] [, DELETE <i>line# 1 – line# 2</i>]</li> </ol>
<b>Example</b>	<p>CHAIN MERGE "PRO1.BAS" , 1000 , ALL</p> <p>Merges the program PRO1.BAS with the current program and executes it from line 1000. All variables remain unchanged.</p>
<b>Description</b>	<p>The CHAIN command reads the program with the specified <i>filename</i> from the memory card and executes the program from the specified <i>line#</i>.</p> <p>If no <i>line#</i> parameter is specified, the program is executed from the beginning.</p> <p>If the command is specified in format 1, the current program is cleared before the specified program is read.</p> <p>If the command is specified in format 2, the current program is merged with the new program. All variables remain unchanged if the ALL parameter is specified.</p>

If *ALL* is not specified, all numeric variables become 0 and character variables become blank (null). Use the *COMMON* command to leave some of the variables unchanged.

If the *DELETE* parameter is used, the program lines between the specified *line# 1* and *line# 2* are deleted before the programs are merged.

Labels can be used instead of numbers for the *line#*, *line# 1*, and *line# 2* parameters.

## **CHANGE**

## **CHANGE scene**

(Command)

**Action** Changes the measuring conditions to the scene setting conditions set in the Menu mode.

**Format** CHANGE *scene#*

**Example** CHANGE 2  
Switch to the scene 2 measuring conditions.

**Description** The CHANGE command refers to the scene setting data previously set in the Menu mode and changes the setting data to the scene specified with the *scene#*.

The following items can be changed:

- Drawing the window graphic (WINDOW)
- Measuring mode for each binary plane (MMODE)
- Camera selection (CAMERA)
- Displayed image (DISP, FILTERIN)
- Window display image (WDISP)
- Filtering (FILTER, FILTDATA)
- Binary level (LEVEL)

However, the following settings can only be made from the Menu mode, they are not possible with the CHANGE command:

- Measurement with multiple camera
- Window displacement compensation
- Shading displacement compensation
- Window measurement items
- Window evaluation conditions

If measurement with multiple cameras is selected in the Menu mode, the camera specified in the Menu mode for the display is used with the CHANGE command.

## **CHDIR**

## **Change DIRectory**

(Command)

**Action** Changes the current directory.

**Format** CHDIR *directory name*

**Example** CHDIR ". . SRC"  
Switches the current directory to directory SRC, which is on the same hierarchical level as the current level (that is, both directories are contained in the same directory on the next level of the hierarchy.)

**Description** Specify the directory name parameter as a character string. Specify the name of a directory existing on the memory card. An error (Path not found) occurs if the specified directory does not exist on the memory card.

To specify an absolute path name for the directory, delimit the directory names with ¥ signs. The total length, including the ¥ signs, must not exceed 127 characters.

Specify the *directory name* as [“.”] to switch the current directory up to the next hierarchy.

**CHR\$****CHaRacter\$**

(Function)

**Action** Determines the character corresponding to a character code.

**Format** CHR\$ (*numeric expression*)

**Example** A\$ = CHR\$ (97)  
Assigns the character “a” corresponding to character code 97 (= 61H) to variable A\$.

**Description** The CHR\$ functions returns the 1-byte character or the control code corresponding to the character code.

Specify the *numeric expression* as an integer between 0 and 255.

The ASV function ASC has the opposite action to the CHR\$ function. ASC returns the decimal character code corresponding to a character.

**CINT****Convert INTeger**

(Function)

**Action** Converts a value to an integer.

**Format** CINT (*numeric expression*)

**Example** SEI% = CINT (A#)  
Converts the value of the double-precision real number A# to an integer and assigns it to the integer variable SEI%.

**Description** The CINT function rounds off the decimal places of the specified *numeric expression* and returns an integer.

An error occurs if the integer returned by the CINT function does not lie between  $-32768$  ( $-2^{16}$ ) and  $32767$  ( $2^{16}-1$ ).

The following type conversion functions are also available:

- CDBL function: Converts a value to a double-precision real number.
- CLNG function: Converts a value to a long integer
- CSNG function: Converts a value to a single-precision real number.

The FIX and INT functions are similar to the CINT function. However, the FIX function simply cuts off (rounds down) the decimal places from the specified value. The INT function also rounds down the specified value, but this function never returns a value larger than the value specified with the numeric expression.

Examples of the actions of the FIX, INT, and CINT function are shown in the table below.

	Specified function	Returned value
Positive numeric expression	FIX (1.7)	1
	INT (1.7)	1
	CINT (1.7)	2
Negative numeric expression	FIX (-1.7)	-1
	INT (-1.7)	-2
	CINT (-1.7)	-2

**CIRCLE****CIRCLE**

(Command)

<b>Action</b>	Draws a circle in VRAM.
<b>Format</b>	CIRCLE <i>X</i> , <i>Y</i> , <i>R</i> , <i>VRAM</i> , [, [[ <i>page#</i> ]] [, [ <i>drawing density or drawing mode</i> ] [, <i>lineart</i> ]]]
<b>Example</b>	CIRCLE 128,255, 25, 2,, 128, 0 Draws a circle on Plane 7 of the window memory with center coordinates (128, 255) and radius (25).
<b>Description</b>	<p>The CIRCLE command draws a circle around the center coordinates (X,Y) with the specified radius (R).</p> <p>Specify the <i>VRAM</i> where the circle is drawn with a number, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i>. The default value is drawing mode, OR.</p> <p>The <i>drawing density</i> parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The <i>drawing density</i> parameter has the following effect when set for the character or graphic memory:</p> <ul style="list-style-type: none"> <li>0 : 0 written to memory</li> <li>!0: 1 written to memory</li> </ul> <p>The <i>drawing mode</i> settings operate as follows.</p> <ul style="list-style-type: none"> <li>OR: The current contents of the image memory ORed with 255 are written to memory.</li> <li>NOT: NOT: 0 is written to memory</li> <li>XOR: XOR: The current contents of the image memory are inverted.</li> </ul> <p>The default value for the <i>drawing mode</i> is OR.</p> <p>Specify with the <i>lineart</i> parameter if the circle is an outline only or filled.</p> <ul style="list-style-type: none"> <li>0: Filled circle</li> <li>1: Circle outline only</li> </ul> <p>The default value is 0 (filled).</p> <p>When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.</p>

**CLEAR****CLEAR**

(Command)

<b>Action</b>	Initializes variables.
<b>Format</b>	CLEAR
<b>Example</b>	CLEAR
<b>Description</b>	The CLEAR command initializes all numeric variables to 0 and all character variables to a blank (null).

**CLNG****Change to LONG integer**

(Function)

<b>Action</b>	Converts a value to a long integer.
<b>Format</b>	CLNG ( <i>numeric expression</i> )
<b>Example</b>	A& = CLNG(B#) Converts the value of the double-precision real number B# to a long integer and assigns it to the long integer variable A&.
<b>Description</b>	The CLNG function rounds off the decimal places of the specified numeric expression and returns a long integer. An error occurs if the long integer returned by the CLNG function does not lie between $-2^{31}$ and $2^{31}-1$ . The following type conversion functions are also available: CDBL function: Converts a value to a double-precision real number. CINT function: Converts a value to an integer. CSNG function: Converts a value to a single-precision real number.

**CLOSE****CLOSE**

(Command)

<b>Action</b>	Closes an open file.
<b>Format</b>	CLOSE [ <i>file#</i> [, <i>file#</i> ] ...]
<b>Example</b>	CLOSE #1 Closes the file which was opened as file# 1.
<b>Description</b>	The CLOSE command closes a file previously opened for data I/O. Specify the same <i>file#</i> used when the file was opened with the OPEN command. After a file is closed with the CLOSE command, data I/O operations on the file are not possible until the file is reopened with the OPEN command. After a file is closed with the CLOSE command, the same <i>file#</i> can be used to open a different file with the OPEN command. Alternatively, the closed file can be reopened with the original <i>file#</i> . Multiple file numbers ( <i>file#</i> ) can be specified to close multiple files simultaneously with a single CLOSE command. All open files are closed if the <i>file#</i> is omitted. Similarly, all files are closed automatically when the END, NEW, or STOP command is executed. When a file opened for data output is closed, all data remaining in the file buffer is written to the file before it is closed. The CLOSE command must be used to ensure this data is correctly written to the file.

**CLS**

**CLeAr Screen**

(Command)

<b>Action</b>	Clears the specified VRAM.
<b>Format</b>	CLS [VRAM [, [page#] [, plane#]]]
<b>Example</b>	CLS 2 , , 3 Clears plane 3 of the window memory.
<b>Description</b>	The CLS command clears the specified VRAM. Specify the VRAM to be cleared with a number, as follows. <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> Omit the <i>page#</i> or set to 0. Specify the plane to be cleared with the <i>plane#</i> parameter. All planes are cleared if this parameter is omitted. When a frame memory is specified, the contents of planes write protected with the MASKBIT command remain unchanged.

**COLOR**

**COLOR**

(Command)

<b>Action</b>	Changes the text display. character attribute.
<b>Format</b>	COLOR <i>attribute code</i>
<b>Example</b>	COLOR 2 Sets the character attribute to overline.
<b>Description</b>	The COLOR command changes the character attribute to the specified attribute. Specify the attribute with a number, as follows. <ul style="list-style-type: none"> <li>0: Clear attribute</li> <li>2: Overline</li> <li>4: Underline</li> <li>7: Reverse</li> </ul>

**COLOR@**

**COLOR@**

(Command)

<b>Action</b>	Changes the character attribute within a rectangular region of the text display.
<b>Format</b>	COLOR@ (X1, Y1) – (X2, Y2) [,attribute code]
<b>Example</b>	COLOR@ (0, 0) – 31, 5), 7 Reverses the characters between line 0, character 0 and line 5, character 31.
<b>Description</b>	The COLOR@ command changes the character attribute to the specified attribute within the rectangular region between the opposing corners specified in character coordinates (X1, Y1) and (X2, Y2). Specify the horizontal character coordinates X1 and X2 as an integer between 0 and 63 and the vertical character coordinates Y1 and Y2 between as an integer between 0 and 24.

Specify the *attribute code* with a number, as follows.

- 0: Clear attribute
- 2: Overline
- 4: Underline
- 7: Reverse

The default value is 7 (reverse).

## COM ON/OFF/STOP                      COMMunication ON/OFF/STOP

(Command)

<b>Action</b>	Enables, disables, or stops interrupts from the RS-232C.
<b>Format</b>	COM ( <i>RS-232C port#</i> ) ON <b>or</b> OFF <b>or</b> STOP
<b>Example</b>	COM (1) ON Permits branching to the interrupt routine when input data is received via the RS-232C channel 0.
<b>Description</b>	Specify the <i>port#</i> as 1 or 2. The default value is 1. The <i>port#</i> is represented by n in the description below.

<u>COM(n) ON</u>	Branches to the interrupt subroutine at the specified line# or label when input data is received via the specified <i>port#</i> . The line# or label must be specified with the ON COM GOSUB statement before the COM(n) ON command is executed. In addition, use of the RS-232C communication port must be declared with the OPEN command before the COM(n) ON command is executed.
------------------	--

Example:

```

10 OPEN "COM:9600N 82N" AS #1
20 ON COM GOSUB *LABEL1
30 COM ON
   ⋮
70 COM OFF
   ⋮
100 *LABEL1
   ⋮
200 RETURN

```

<u>COM (n) OFF</u>	Disables branching to the interrupt routine when input data is received via the specified <i>port#</i> .
<u>COM (n) STOP</u>	Temporarily stops operation when input data is received via the specified <i>port#</i> . Operation does not branch to the interrupt subroutine. Operation will branch to the interrupt subroutine after the COM (n) ON command is executed.

**Note** The COM (n) OFF status must be set before the program execution ends.

## COMMON

## COMMON

(Command)

<b>Action</b>	Transfers variables to a program linked with the CHAIN command.
<b>Format</b>	COMMON <i>variable name</i> [, <i>variable name</i> ...]
<b>Example</b>	COMMON X, Y, A\$, B Transfers variables X, Y, A\$ and B to a specified program in the CHAIN command.



**Description** The COMMON command transfers variables to another program. The COMMON command is used as a pair with the CHAIN command. The COMMON statement must be declared before the CHAIN statement, that is, at the start of the program.

Variables must not be duplicated in a single COMMON command.

Add parentheses to declare array variables, e.g., B( ).

Use the ALL parameter in the CHAIN statement to transfer all variables to the linked program.

Example:

```

10 DIM B(100), C(10), D(10)
    :
    :
50 CHAIN "TEST.BAS" , , ALL
    
```

**CONSOLE**

**CONSOLE**

(Command)

**Action** Sets the text display mode.

**Format** CONSOLE [[*scroll start line*] [, [*number of scroll lines*] [, [*function key display switch*] [, *character mode*]]]]

**Example** CONSOLE , , , 1

Sets the text display mode to the graphic character mode and hold all other parameter settings.

**Description** The CONSOLE command sets the text display mode.

The area designated by the scroll start line and number of scroll lines parameters. The screen clear operation acts on this specified scroll area. The default value for the scroll start line is 0. If the number of scroll lines is not specified, the previous setting is maintained.

The function key display switch setting specifies whether the function key menu is displayed on the bottom line of the screen. The function key menu is not displayed if this parameter is set to 0. If the function key display switch is not specified, the previous setting is maintained.

Set the character mode to 1 to select the graphic character mode. This mode permits display of the graphic characters (character codes &H80 to &H9F and &HE0 to &HF7) but disables the Japanese character display.

Set the character mode to 0 to enable the Japanese character display but disable display of the graphic characters. If no setting is specified, the previous setting is maintained.

**CONT**

**CONTInue**

(Command)

**Action** Continues execution of a stopped user program.

**Format** CONT

**Example** CONT

Restarts program execution.

**Description** The CONT command is a direct command to continue execution of a user program stopped by pressing the STOP or CTRL+C Keys or by executing the STOP command.

Operations such as printing the variable names in the direct mode are possible while the program is stopped. However, a program cannot be continued if it was modified while execution was stopped.

**COS**

**COSine**

(Function)

<b>Action</b>	Determines the cosine of an angle.
<b>Format</b>	COS ( <i>numeric expression</i> )
<b>Example</b>	A! = COS(60*3.14159/180) Assigns the cosine of 60% (0.50000) to variable A!.
<b>Description</b>	Returns the cosine as a value between -1 and 1. The angle in the <i>numeric expression</i> is set in radians. The <i>numeric expression</i> may be in the form of an integer, long integer, or a single or double-precision real number. The COS function always returns a double-precision real number.

**CSNG**

**Convert SiNGle**

(Function)

<b>Action</b>	Converts a value to a single-precision value.
<b>Format</b>	CSNG ( <i>numeric expression</i> )
<b>Example</b>	SNG! = CSNG (A#) Converts the value of the double-precision real number A# to a single-precision real number and assigns it to the single-precision real number variable SNG!.
<b>Description</b>	The CSNG function returns the specified <i>numeric expression</i> converted to a single-precision real number. The CSNG function converts values to a single-precision real value between -1.70141E+38 and -1.70141E+38. An error occurs if the value lies outside this range. The following type conversion functions are also available: CDBL function: Converts a value to a double-precision real number. CLNG function: Converts a value to a long integer CINT function: Converts a value to an integer.

**CSRLIN**

**CurSor LiNE**

(Function)

<b>Action</b>	The CSRLIN function determines the number of the line where the character cursor is positioned.
<b>Format</b>	CSRLIN
<b>Example</b>	LN% = CSRLIN Assigns the current cursor line position to the variable LN%.
<b>Description</b>	Returns the line position of the character cursor (the Y-axis character coordinate). The value returned is between 0 and 24. Use the POS function to determine the cursor character position along the line.

**CURSOR**

**CURSOR**

(Command)

<b>Action</b>	Draws the cross cursor in VRAM.
<b>Format</b>	CURSOR X, Y, <i>angle</i> , VRAM [, [ <i>page#</i> ][, <i>drawing density or drawing mode</i> ]]
<b>Example</b>	CURSOR 200, 250, 45, 1 Draws a cross cursor at an angle of 45% in the graphic memory at coordinate position (200,250).

<b>Description</b>	<p>The CURSOR command draws a cursor at coordinate position (X,Y) at the specified angle.</p> <p>The angle parameter specifies the tilt angle of the cross cursor.</p> <p>Specify the VRAM where the cursor is drawn with a number, as follows.</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i>. The default value is <i>drawing mode</i>, OR.</p> <p>The <i>drawing density</i> parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The <i>drawing density</i> parameter has the following effect when set for the character or graphic memory:</p> <ul style="list-style-type: none"> <li>0 : 0 written to memory</li> <li>!0: 1 written to memory</li> </ul> <p>The <i>drawing mode</i> settings operate as follows.</p> <ul style="list-style-type: none"> <li>OR: The current contents of the image memory ORed with 255 are written to memory.</li> <li>NOT: NOT: 0 is written to memory</li> <li>XOR: XOR: The current contents of the image memory are inverted.</li> </ul> <p>The default value for the drawing mode is OR.</p> <p>When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.</p>
--------------------	---

**CVD****ConVert to Double**

(Function)

<b>Action</b>	Converts an 8-byte character string to a double-precision real number.
<b>Format</b>	CVD ( <i>8-byte character string</i> )
<b>Example</b>	<p>C# = CVD(C\$)</p> <p>Converts a character-type numeric data variable C\$ created with the MKD\$ function to a double-precision real number and assigns it to variable C#.</p>
<b>Description</b>	<p>The CVD function returns the value of an <i>8-byte character string</i> converted to a double-precision real number.</p> <p>Because numeric data cannot be handled in a random access file, a a double-precision real number must be converted to a character-type numeric data variable (character string) with the MKD\$ function before it is written to a random access file. The CVD function converts character-type numeric data read from a random access file back to a double-precision real number.</p> <p>The CVD function has the opposite action to the MKD\$ function. The MKD\$ function converts numeric data to a character string.</p> <p>Character-type numeric data is read from a random access file with the GET# command to the variable areas defined with the FIELD# command in the file buffer.</p>

**CVI****ConVert to Integer**

(Function)

<b>Action</b>	Converts a 2-byte character string to an integer.
<b>Format</b>	CVI ( <i>2-byte character string</i> )
<b>Example</b>	A% = CVI(A\$) Converts a character-type numeric data variable A\$ created with the MKI\$ function to an integer and assigns it to variable A%.
<b>Description</b>	The CVI function returns the value of a <i>2-byte character string</i> converted to an integer. Because numeric data cannot be handled in a random access file, an integer must be converted to a character-type numeric data variable (character string) with the MKI\$ function before it is written to a random access file. The CVI function converts character-type numeric data read from a random access file back to an integer. The CVI function has the opposite action to the MKI\$ function. The MKI\$ function converts numeric data to a character string. Character-type numeric data is read from a random access file with the GET# command to the variable areas defined with the FIELD# command in the file buffer.

**CVL****ConVert Long integer**

(Function)

<b>Action</b>	Converts a 4-byte character string to a long integer.
<b>Format</b>	CVL ( <i>4-byte character string</i> )
<b>Example</b>	A& = CVL(A\$) Converts a character-type numeric data variable A\$ created with the MKL\$ function to a long integer and assigns it to variable A&.
<b>Description</b>	The CVL function returns the value of a <i>4-byte character string</i> converted to a long integer. Because numeric data cannot be handled in a random access file, a long integer must be converted to a character-type numeric data variable (character string) with the MKL\$ function before it is written to a random access file. The CVL function converts character-type numeric data read from a random access file back to long integer data. The CVL function has the opposite action to the MKL\$ function. The MKL\$ function converts numeric data to a character string. Character-type numeric data is read from a random access file with the GET# command to the variable areas defined with the FIELD# command in the file buffer.

**CVS****ConVert to Single**

(Function)

<b>Action</b>	Converts an 4-byte character string to a single-precision real number.
<b>Format</b>	CVS ( <i>4-byte character string</i> )
<b>Example</b>	B! = CVS(B\$) Converts a character-type numeric data variable B\$ created with the MKS\$ function to a single-precision real number and assigns it to variable B!.

**Description** The CVS function returns the value of an 4-byte character string converted to a single-precision real number.

Because numeric data cannot be handled in a random access file, a single-precision real number must be converted to a character-type numeric data variable (character string) with the MKS\$ function before it is written to a random access file. The CVS function converts character-type numeric data read from a random access file back to a single-precision real number.

The CVS function has the opposite action to the MKS\$ function. The MKS\$ function converts numeric data to a character string.

Character-type numeric data is read from a random access file with the GET# command to the variable areas defined with the FIELD# command in the file buffer.

**DATA**

**DATA**

(Command)

**Action** Defines the integer constants and character constants read with the READ command.

**Format** DATA *constant* [, *constant* [, *constant*...]]

**Example**

DATA 10, 20, 30, 40  
 Defines the integer constants 10, 20, 30, and 40 as decimal data.

DATA 5.5, "OVL", 200  
 DATA 5.5, OVL, 200  
 Defines the real number 5.5, character string "OVL," and integer 200 as data.

**Description** The DATA statement is not executable. It can be declared anywhere in the program.

The *constants* (numeric and character) are delimited by commas (,).

A *constant* cannot be defined as a constant expression, such as 10\*2.

When one of the character strings described below is declared as a *constant*, it must be enclosed in double quotations (").

- A character string containing a meaningful symbol such as a comma (,), colon (:), or period (.)
- A character string starting or ending with a meaningful blank.

The constants defined with the DATA command are read sequentially to the variables defined with the READ command.

Normally the DATA statement constants and the READ statement variables have the same format. However, it is possible to specify character variables for the READ statement which are read as the character-type numeric variables in the corresponding DATA statement.

DATA statement constant format	READ statement variable format	Results of executing READ statement
Numeric constant	Numeric variable	Read as a normal number
Numeric constant	Character variable	Number read as a character string
Character constant	Character variable	Read as a normal character string
Character constant	Numeric variable	Error

A RESTORE command before the READ command specifies the line containing the DATA statement to read. If the DATA statement is not specified by a

RESTORE command, the position of the data read depends on the execution status of the READ command.

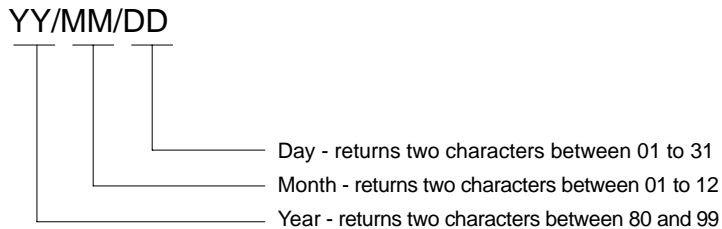
Execution status of READ command	Data to be read
First execution in program	Sequentially reads the data beginning with the first data statement in the program.
Second and succeeding executions	Continues reading the data sequentially after all remaining data from the first execution has been read.

## DATE\$

## DATE\$

(Command-Function)

- Action** Displays and sets the date in the internal clock.
- Format** Format 1 DATE\$  
Format 2 DATE\$ = "yy/mm/dd"
- Example** PRINT DATE\$  
Displays the date returned in DATE\$.  
DATE\$ = "92/11/10"  
Sets the date to 10 November, 1992.
- Description** Reads the date from the F300.  
The F300 has no internal clock. The clock is set to 00/00/00 on 80/01/01 when the power supply is turned on. The time elapsed since the power was turned on is counted. Therefore, it is not possible to read the correct date with the DATE\$ function until the clock has been set.  
A character string with the year, month, and day (yy/mm/dd) delimited by slashes is returned when the DATE\$ function is executed. The format is shown below. The last two digits of the year are returned.



## DEF FN

## DEFine Function

(Command)

- Action** Defines a user function.
- Format** DEF FN *name* [(*parameter* [, *parameter*...])] = *expression*
- Example** DEF FNX% (X%) = X% \* X%  
Defines a function to square the parameter.
- Description** The DEF function defines a function and its name.  
Function names are subject to the same restrictions as variable names.  
The *parameters* have the same names as the variables used inside the function. These variable names are used only for evaluation of the expression inside the function. The same variable names can be used elsewhere inside the program.  
The expression declares the operation of the function. It must not exceed one line.

The function is called in the format: FN *name* (*variable*). It is not necessary for the variable name used to match the name used in the expression declaring the operation of the function, however the variable format must be the same. The function must be declared before it can be called.

## DEF FN...END DEF      DEFine FuNction END DEFine

(Command)

<b>Action</b>	Defines a user function block.
<b>Format</b>	DEF FN <i>name</i> [( <i>parameter</i> [, <i>parameter</i> ...])] <i>statement in DEF FN block</i> END DEF
<b>Example</b>	DEF FNX% (X%) : : FNX% = A% + X% END DEF Defines function X.
<b>Description</b>	Defines a user function block. The function names must conform to restrictions covering variable names. The <i>parameters</i> have the same names as the variables used inside the function. These variable names are used only for evaluation of the expression inside the function. The same variable names can be used elsewhere inside the program. It is possible to describe operation expressions as statements in more than one line in a DEF FN statement block. Use the format FN <i>name</i> ( <i>variable</i> ) to have access to a function. A DEF FN statement must exist before the above format. No further DEF FN ... END DEF declaration can be nested inside a DEF FN statement block. The GOTO command must not be used to jump into or out of a DEF FN statement block. Use the EXIT DEF command to get out of a DEF FN statement block.

## DEFDBL

## DEFine DouBLe

(Command)

<b>Action</b>	Declares variables as double-precision real number variables.
<b>Format</b>	DEFDBL <i>character range</i> [, <i>character range</i> ...]
<b>Example</b>	DEFDBL E-H, Z Declares all variables starting with the letters E, F, G, H, and Z as double-precision real numbers.
<b>Description</b>	Declares variables starting with characters in the specified <i>character range</i> as double-precision real number variables. Only single characters can be specified in the <i>character range</i> . Specify multiple characters using the minus (-) sign. The type statements (% , & , ! , # , \$) take priority over type declarations with this command.

## DEFINT

## DEFine INTeger

(Command)

<b>Action</b>	Declares variables as integers.
<b>Format</b>	DEFINT <i>character range</i> [, <i>character range</i> ...]
<b>Example</b>	DEFINT A, F-I Declares all variables starting with the letters A, F, G, H, and I as integers.

<b>Description</b>	Declares variables starting with characters in the specified <i>character range</i> as integer variables. Only single characters can be specified in the <i>character range</i> . Specify multiple characters using the minus (–) sign. The type statements (% , & , ! , # , \$) take priority over type declarations with this command.
--------------------	--

**DEF LNG****DEFine LoNG**

(Command)

<b>Action</b>	Declares variables as long integers.
<b>Format</b>	DEF LNG <i>character range</i> [, <i>character range</i> ...]
<b>Example</b>	DEF LNG B , C–F Declares all variables starting with the letters B, C, D, E, and F as long integers.
<b>Description</b>	Declares variables starting with characters in the specified <i>character range</i> as long integer variables. Only single characters can be specified in the <i>character range</i> . Specify multiple characters using the minus (–) sign. The type statements (% , & , ! , # , \$) take priority over type declarations with this command.

**DEF SNG****DEFine SiNGle**

(Command)

<b>Action</b>	Declares variables as single-precision real number variables.
<b>Format</b>	DEF SNG <i>character range</i> [, <i>character range</i> ...]
<b>Example</b>	DEF SNG A , B–D Declares all variables starting with the letters A, B, C, and D as single-precision real numbers.
<b>Description</b>	Declares variables starting with characters in the specified <i>character range</i> as single-precision real number variables. Only single characters can be specified in the <i>character range</i> . Specify multiple characters using the minus (–) sign. The type statements (% , & , ! , # , \$) take priority over type declarations with this command.

**DEF STR****DEFine STRing**

(Command)

<b>Action</b>	Declares variables as character variables.
<b>Format</b>	DEF STR <i>character range</i> [, <i>character range</i> ...]
<b>Example</b>	DEF STR A , B–D Declares all variables starting with the letters A, B, C, and D as character strings.
<b>Description</b>	Declares variables starting with characters in the specified <i>character range</i> as character string variables. Only single characters can be specified in the <i>character range</i> . Specify multiple characters using the minus (–) sign. The type statements (% , & , ! , # , \$) take priority over type declarations with this command.



**DELETE****DELETE**

(Command)

<b>Action</b>	Deletes all lines in a specified range of the program.
<b>Format</b>	DELETE [ <i>line# 1</i> ] [ <i>-line# 2</i> ]
<b>Example</b>	DELETE 100-190 Deletes all lines from 100 to 190.
<b>Description</b>	Deletes all lines between <i>line# 1</i> and <i>line# 2</i> . Only <i>line# 1</i> is deleted if <i>line# 2</i> is not specified. Lines from the start of the program to <i>line# 2</i> are deleted if <i>line# 1</i> is not specified. Use a period (.) instead of <i>line# 1</i> or <i>line# 2</i> to specify the current program line. The current line is indicated by the pointer. The current line is the last line input during program creation or the last line displayed with the LIST command. It is not possible to omit both the <i>line# 1</i> and <i>line# 2</i> parameters.

**DEVICE****DEVICE**

(Command)

<b>Action</b>	Specifies the standard OVL input and output devices.
<b>Format</b>	DEVICE ["KYBD:" or "COM:"], ["SCRN:" or "COM:"]
<b>Example</b>	DEVICE "COM: " , "COM: " Specifies program input and output through the RS-232C port.
<b>Description</b>	The input device can be selected as the keyboard or RS-232C port. The output device can be selected as the video monitor or the RS-232C port. The first parameter specifies the input device, as one of the following. KYBD: Keyboard COM: RS-232C Input from the keyboard is disabled if the input device is set to the RS-232C port. The second parameter specifies the input device, as one of the following. SCRN: Video monitor COM: RS-232C Subsequent characters are output to the RS-232C port if the output device is set to RS-232C. Previous settings remain unchanged if both parameters are omitted.

**DIM****DIMension**

(Command)

<b>Action</b>	Defines an array variable.
<b>Format</b>	DIM <i>variable name</i> ( <i>qualifier max. value</i> [, <i>qualifier max. value ...</i> ]) [, <i>variable name</i> ( <i>qualifier max. value</i> [, <i>qualifier max. value...</i> ])]

<b>Example</b>	<p>DIM A\$(50)</p> <p>Defines a one-dimensional character array variable A\$(50).</p> <p>Defines a two-dimensional single-precision real number array variable B\$(10,3).</p>
<b>Description</b>	<p>Declares the variable format, number of dimensions, and <i>qualifier maximum values</i> of an array variable and assigns the array variable to an area of memory.</p> <p>The array variable format is specified with the type statements (% , ! , # , \$ , &amp;). The default type if the type statement is omitted is a single-precision real number variable.</p> <p>The number of <i>qualifier maximum values</i> indicates the number of dimensions of the array. If multiple dimensions are specified, the <i>qualifier maximum values</i> are delimited by commas (,).</p> <p>The <i>qualifier maximum value</i> parameters specify the maximum value a qualifier can have. The qualifier minimum value can be specified as 0 or 1 with the OPTION BASE statement.</p> <p>The default value is 0 if no OPTION BASE statement is specified. Note that executing the command DIM A(20) if no OPTION BASE statement is specified in the program results in an array with elements from 0 to 20, that is, 21 elements.</p>

## DIN

## Data IN

(Function)

<b>Action</b>	Reads the status of the Terminal Block Unit or Parallel I/O Unit input port.
<b>Format</b>	DIN ( <i>bit address</i> [, <i>size</i> ])
<b>Example</b>	<p>A = DIN (2, 8)</p> <p>Reads the status of the eight bits 2 to 9 from the Terminal Block Unit or Parallel I/O Unit and assigns the bits to variable A.</p>
<b>Description</b>	<p>The DIN function reads the data input to the Terminal Block Unit or Parallel I/O Unit input ports.</p> <p>If both Terminal Block Units and Parallel I/O Units are connected, the DIN function does not differentiate between the two. The maximum number of input bits becomes the total number of the input bits for all Units.</p> <p>Specify the number of input data bits with the size parameter. The default value is 8. Specify the data input start position with the bit address parameter.</p> <p>Data is input as integers with + or – signs.</p>

## DISPLAY

## DISPLAY

(Command)

<b>Action</b>	Sets the display image.
<b>Format</b>	DISPLAY <i>display image</i> [, <i>image pathway</i> ]
<b>Example</b>	<p>DISPLAY 31</p> <p>Displays all images.</p>

<b>Description</b>	<p>The DISPLAY command specifies the image displayed on the video monitor as a display number between 0 and 31.</p> <p>The decimal number between 0 and 31 set for the <i>display image</i> represents a binary number. The image corresponding to a bit set to 1 is displayed. The relationship between the bits and displayed images is shown below. Normally leave bit 0 set to 1.</p> <p>Bit 0: Camera image          Bit 1: Paint/pattern matching window memory image          Bit 2: Window memory image          Bit 3: Graphic memory image          Bit 4: Character memory image</p> <p>The <i>image pathway</i> parameter specifies the image bus taken by the displayed image. The default value is 2.</p> <p>0: Binary image and raw image .. Image bus 0          1: Binary image ... Image bus 1, raw image ... Image bus 0          2: Binary image ... Image bus 0, raw image ... Image bus 1          3: Binary image and raw image .. Image bus 1</p>
--------------------	---

**DO-LOOP REPEAT****DO-LOOP REPEAT**

(Command)

<b>Action</b>	Repeats the statement between DO and LOOP the specified number of repetitions.
<b>Format</b>	<p>DO</p> <p style="padding-left: 40px;"><i>Statements in DO block</i></p> <p>LOOP REPEAT <i>number of times</i></p>
<b>Example</b>	<pre>DO PRINT I LOOP REPEAT 20</pre> <p>Displays variable I 21 times.</p>
<b>Description</b>	<p>The DO-LOOP REPEAT command executes the statement between DO and LOOP the specified <i>number of times</i>.</p> <p>The <i>statements in the DO block</i> are executed the <i>number of times</i> specified by the (<i>number of times</i> parameter + 1).</p> <p>The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.</p>

**DO-LOOP UNTIL****DO-LOOP UNTIL**

(Command)

<b>Action</b>	Repeats the statements between DO and LOOP until the condition is fulfilled.
<b>Format</b>	<p>DO</p> <p style="padding-left: 40px;"><i>Statements in DO block</i></p> <p>LOOP UNTIL <i>conditional expression</i></p>
<b>Example</b>	<pre>I = 20 DO PRINT I I = I + 1 LOOP UNTIL I &gt; 29</pre> <p>Displays the integers from 20 to 29.</p>

<b>Description</b>	<p>The DO-LOOP UNTIL command executes the statement between DO and LOOP until the condition is fulfilled.</p> <p>The statements in the DO block are executed repeatedly while the conditional expression is false.</p> <p>The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.</p>
--------------------	--

**DO-LOOP WHILE****DO-LOOP WHILE**

(Command)

<b>Action</b>	Repeats the statements between DO and LOOP while the condition is fulfilled.
<b>Format</b>	<p>DO</p> <p style="padding-left: 40px;"><i>Statements in DO block</i></p> <p>LOOP WHILE <i>logical expression</i></p>
<b>Example</b>	<pre>I=0 DO PRINT I I=I+1 LOOP WHILE I&lt;10</pre> <p>Displays the integers from 0 to 9.</p>
<b>Description</b>	<p>The DO-LOOP WHILE command executes the statement between DO and LOOP while the condition is fulfilled.</p> <p>The statements in the DO block are executed repeatedly while the logical expression is true.</p> <p>The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.</p>

**DO REPEAT-LOOP****DO REPEAT-LOOP**

(Command)

<b>Action</b>	Repeats the statements between DO and LOOP the specified number of times.
<b>Format</b>	<p>DO REPEAT <i>number of times</i></p> <p style="padding-left: 40px;"><i>Statements in DO block</i></p> <p>LOOP</p>
<b>Example</b>	<pre>DO REPEAT 20 PRINT I LOOP</pre> <p>Displays variable I 20 times.</p>
<b>Description</b>	<p>The DO REPEAT and LOOP command executes the statement between DO and LOOP the specified <i>number of times</i>.</p> <p>The statements in the DO block are executed the specified number of times.</p> <p>The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.</p>

**DO UNTIL-LOOP****DO UNTIL-LOOP**

(Command)

<b>Action</b>	Repeats the statements between DO and LOOP until the condition is fulfilled.
<b>Format</b>	DO UNTIL <i>logical expression</i> <i>Statements in DO block</i> LOOP
<b>Example</b>	I=25 DO UNTIL I <20 PRINT I I=I-1 LOOP Displays the integers from 25 to 20.
<b>Description</b>	The DO UNTIL-LOOP command executes the statement between DO and LOOP until the condition is fulfilled. The statements in the DO block are executed repeatedly while the logical expression is false. The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.

**DO WHILE-LOOP****DO WHILE-LOOP**

(Command)

<b>Action</b>	Repeats the statements between DO and LOOP while the condition is fulfilled.
<b>Format</b>	DO WHILE <i>logical expression</i> <i>Statements in DO block</i> LOOP
<b>Example</b>	I=0 DO WHILE I <10 PRINT I I=I+1 LOOP Displays the integers from 0 to 9.
<b>Description</b>	The DO WHILE-LOOP command executes the statement between DO and LOOP while the condition is fulfilled. The statements in the DO block are executed repeatedly while the logical expression is true. The GOTO command must not be used to jump into or out of a DO block. Use the EXIT DO command to get out of a DO block.

**DOUT****Data OUTput**

(Command)

<b>Action</b>	Outputs data to a Terminal Block Unit or Parallel I/O Unit.
<b>Format</b>	DOUT <i>output data</i> [, [ <i>bit address</i> ] [, <i>size</i> ]]
<b>Example</b>	DOUT 123,16,8 Outputs the data 123 as eight bits from bit 16 of the Terminal Block Unit or Parallel I/O Unit.

**Description** The DOUT function outputs the data specified with the *output data* parameter to the output port of the Terminal Block Unit or Parallel I/O Unit.

If both Terminal Block Units and Parallel I/O Units are connected, the DOUT function does not differentiate between the two. The maximum number of output bits becomes the total number of the output bits for all units.

Specify the data output start position (0 – (max. no. of output bits –1)) with the *bit address* parameter. The default value is 0.

Specify the number of data bits to define the output data with the *size* parameter. The default value is 16.

**DSA**

**Data Set Acknowledge**

(Function)

**Action** Reads the status of the DSA signal.

**Format** DSA

**Example** A=DSA

Assigns the status of the Terminal Block Unit or Parallel I/O Unit DSA signal to variable A.

**Description** The DSA function reads the status of the Terminal Block Unit or Parallel I/O Unit DSA signal, and returns a value as follows:

- 1: DSA signal is ON.
- 0: DSA signal is OFF.

When multiple I/O units are used simultaneously, this function reads the DSA signal status from the unit with the lowest slot address.

**DSKF**

**DiSK Function**

(Function)

**Action** Determines the free space remaining in the memory card.

**Format** DSKF (*drive*)

**Example** J=DSKF( "C: " )

Determines the free space remaining in the memory card.

**Description** Determines the number of free bytes remaining in the disk specified by the drive parameter.

The memory card interface is Drive C.

**EDIT**

**EDIT**

(Command)

**Action** Selects the mode to edit lines of the program.

**Format** EDIT *line number or* .

**Example** EDIT 100

Displays line#1000 and selects the Edit mode.

**Description** Displays the line with the specified line number and selects the Edit mode.

In the Edit mode, the lines are edited with the ROLL-UP, ROLL-DOWN, and other Keys. Use a period (.) to specify the current program line.

The current line is indicated by the pointer. The current line is the last line input during program creation or the last line displayed with the LIST command.

**ELLIPSE****ELLIPSE**

(Command)

<b>Action</b>	Draws an ellipse in VRAM.
<b>Format</b>	ELLIPSE X, Y, XR, YR, VRAM[, [page#], [drawing density or drawing mode] [, lineart]]
<b>Example</b>	ELLIPSE 128,255,90,35,2,,128,0 Draws a filled ellipse on Plane 7 of the window memory with center coordinates (128,255), horizontal axis (9), and vertical axis (35).
<b>Description</b>	<p>The ELLIPSE command draws an ellipse around the center coordinates (X,Y) with the specified horizontal axis (XR), and vertical axis (YR).</p> <p>Specify the VRAM where the ellipse is drawn with a number, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the page# or set to 0.</p> <p>Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i>. The default value is <i>drawing mode</i>, OR.</p> <p>The <i>drawing density</i> parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The <i>drawing density</i> parameter has the following effect when set for the character or graphic memory:</p> <ul style="list-style-type: none"> <li>0 : 0 written to memory</li> <li>!0: 1 written to memory</li> </ul> <p>The <i>drawing mode</i> settings operate as follows:</p> <ul style="list-style-type: none"> <li>OR: The current contents of the image memory ORed with 255 are written to memory.</li> <li>NOT: 0 is written to memory</li> <li>XOR: The current contents of the image memory are inverted.</li> </ul> <p>The default value for the <i>drawing mode</i> is OR.</p> <p>Specify with the <i>lineart</i> parameter if the ellipse is an outline only or filled.</p> <ul style="list-style-type: none"> <li>0: Filled ellipse</li> <li>1: Ellipse outline only</li> </ul> <p>The default value is 0 (filled).</p> <p>When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.</p>

**END****END**

(Command)

<b>Action</b>	Stops program execution.
<b>Format</b>	END
<b>Example</b>	IF A\$="FINISH" THEN END Stops program execution if A\$ contains the character string "FINISH".
<b>Description</b>	<p>Stops program execution.</p> <p>All files opened during program execution are closed when the END command is executed. Even if no END command is executed, all files are closed and program execution stops when the last line of the program has been executed.</p>

**ENHANCE****ENHANCE**

(Command)

<b>Action</b>	Creates LUT data for contrast modification from histogram array data.
<b>Format</b>	ENHANCE <i>array name, modification array name</i>
<b>Example</b>	ENHANCE A, E Converts the histogram data in array A to contrast-modified histogram data and stores it in array E.
<b>Description</b>	The ENHANCE command creates the original contrast-modified histogram from the histogram data obtained with the HISTGRAM command. This contrast-modified histogram is set into a LUT (look-up table) for binary conversion with the SETLUT command. After the camera image is selected with the FILTERIN command and the image is input to the video memory with the VIDEOIN command, the contrast of the image input from the camera is improved before the image is input to the image memory. Use the the <i>array name</i> parameter to specify the name of the array containing the histogram obtained with the HISTGRAM command. Specify the name of the array to store the converted data with the <i>modification array name</i> parameter. The array to store the modified data must contain at least 256 elements.

**EOF****End Of File**

(Function)

<b>Action</b>	Determines if the file is at the end of the file.
<b>Format</b>	EOF ( <i>file#</i> )
<b>Example</b>	IF EOF(2) THEN CLOSE #2 Closes file #2 during I/O execution if the EOF (end of file) is detected in file #2.
<b>Description</b>	The EOF function can be used for sequential access files and communication ports. Sequential access files on the memory card must be opened in the INPUT mode. The EOF function returns -1 (true) if no readable data remains in the file or 0 (false) if readable data remains in the file. Specify the number of the file opened with the OPEN command with the <i>file#</i> parameter.

**ERASE****ERASE**

(Command)

<b>Action</b>	Deletes an array defined with the DIM command.
<b>Format</b>	ERASE <i>array name</i> [, <i>array name...</i> ]
<b>Example</b>	ERASE X, Y1\$ Deletes arrays X and Y1\$.
<b>Description</b>	The DIM command deletes the array with the name specified by the <i>array name</i> parameter. The amount of memory equivalent to the size of the file is then available to store variables. After an array is deleted, another array with the same name can be created. This allows the size of an array to be changed. A Duplicate Definition error occurs if an array with the same name as an existing array is created without first deleting the original array with the ERASE command.



**ERL** **Error Line**

(Function)

<b>Action</b>	Determines the line number where an error occurred.
<b>Format</b>	ERL
<b>Example</b>	L%=ERL Assigns the line number where the error occurred to the variable L%.
<b>Description</b>	The ERL function returns the line number where an error occurred. The ERL function is normally used inside an error processing routine in combination with program flow control statements. It is used for flow control in error recovery processing and for recovery after the error is reset.

**ERR** **ERRor code**

(Function)

<b>Action</b>	Determines the error code after an error occurs.
<b>Format</b>	ERR
<b>Example</b>	C%=ERR Assigns the error code to the variable C%.
<b>Description</b>	The ERR function returns the error code. The ERR function is normally used inside an error processing routine in combination with program flow control statements. It is used for flow control in error recovery processing and for recovery after the error is reset.

**ERRMSG** **ERRor MeSsaGe**

(Command)

<b>Action</b>	Defines the operation when an error occurs.
<b>Format</b>	ERRMSG [ <i>message</i> ] [, [ <i>buzzer</i> ] [, <i>error signal control</i> ]]
<b>Example</b>	ERRMSG , 1 The buzzer sounds and an error message is displayed when error occurs.
<b>Description</b>	The ERRMSG command defines the message format, buzzer alarm, and error signal control method when an error occurs. Set the <i>message</i> parameter to 0 to specify a Japanese error message or to 1 to specify an English error message. Set the <i>buzzer</i> parameter to 0 to stop buzzer operation when an error occurs. Set the <i>buzzer</i> parameter to 1 to sound the buzzer when an error message is displayed. If the <i>error signal control</i> parameter is set to 0, the error signal does not turn on when an error occurs. Set the <i>error signal control</i> parameter to 1, to turn the error signal on when an error occurs. The previous setting remains unchanged if the <i>buzzer</i> or <i>error signal control</i> parameter is omitted.

**ERROR** **ERROR**

(Command)

<b>Action</b>	Generates a pseudo-error.
<b>Format</b>	ERROR ( <i>numeric expression</i> )
<b>Example</b>	ERROR 2 Generates error code 2: Syntax error.

<b>Description</b>	<p>The ERROR command generates the error specified with the numeric expression. When an error is generated which corresponds to a predefined system error code, program execution is interrupted and the appropriate error message is displayed.</p> <p>Specify the numeric expression as an integer between 0 and 255. This range of integers includes the system error codes.</p> <p>Refer to the Error Message Table on page 213 for details of the error codes.</p>
--------------------	---

**ERROUT****ERRoR OUT**

(Command)

<b>Action</b>	Controls the error output.
<b>Format</b>	ERROUT <i>0 or 1</i>
<b>Example</b>	<p>ERROUT 1</p> <p>Turns ON the power supply unit ERROR signal.</p>
<b>Description</b>	The ERROUT command controls the ON/OFF status of the power supply unit ERROR signal. Set to 0 to turn the ERROR signal OFF or set to 1 to turn the signal ON.

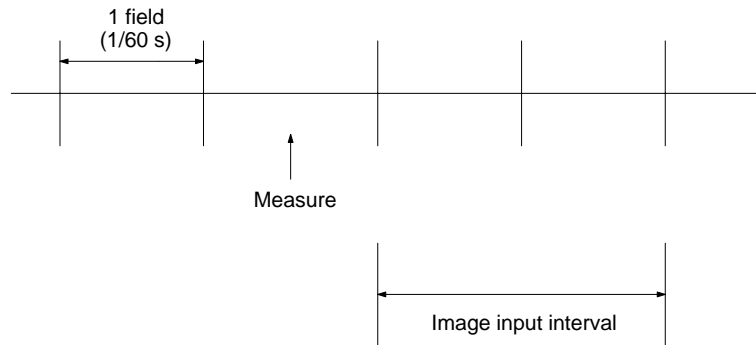
**EVENTIN****EVENT IN**

(Command)

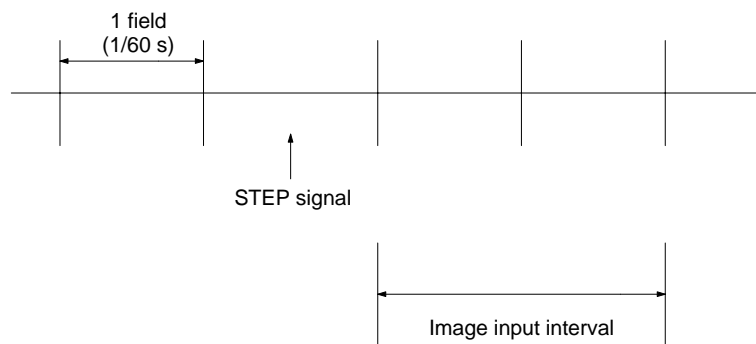
<b>Action</b>	Sets the image input mode for input to the image memory due to event synchronization.
<b>Format</b>	EVENTIN [ <i>input mode</i> [, [ <i>page#</i> ] [, <i>input path</i> ]]]
<b>Example</b>	<p>EVENTIN 2</p> <p>The image is input to page 0 of the image memory when the STEP signal is input.</p>
<b>Description</b>	<p>The EVENTIN command sets the mode to input the image to the input memory in synchronization with the STEP signal or MEASURE command.</p> <p>Set the <i>input method</i> parameter to one of the following values to specify the image input method.</p> <ul style="list-style-type: none"> <li>0: No image input</li> <li>1: Input image synchronized with MEASURE command execution.</li> <li>2: Image input synchronized with the STEP signal.</li> </ul> <p>The default value is 0.</p> <p>Omit the <i>page#</i> or set to 0.</p> <p>Inputting the image to the image memory in synchronization with the MEASURE command allows comparatively high-speed measurements.</p> <p>The image can be successfully input to the image memory with the strobe by synchronizing the image input with the STEP signal in combination with the FLASH command.</p> <p>The <i>input path</i> parameter specifies the bus to input the image. The default value is 0.</p> <ul style="list-style-type: none"> <li>0: Image bus 1</li> <li>1: Image bus 0</li> </ul> <p>The EVENTIN command must be executed each time before executing the MEASURE command or the STEP signal is input.</p>

The read timing for each input mode is shown below.

### Input Mode 1:



### Input Mode 2:



## EXIT DEF/DO/FOR/SUB

## EXIT DEF/DO/FOR/SUB

(Command)

<b>Action</b>	Exits a control block.
<b>Format</b>	Format 1: EXIT DEF Format 2: EXIT DO Format 3: EXIT FOR Format 4: EXIT SUB
<b>Example</b>	<pre>FOR I=0 TO A .   IF I&gt;MAX THEN     PRINT "ERROR"     EXIT FOR   END IF . . NEXT</pre> <p>Operation leaves the FOR-NEXT control loop if the value of variable I exceeds the value of the variable MAX.</p>
<b>Description</b>	Format 1: Exits a user function. Format 2: Exits a DO-LOOP loop. Format 3: Exits a FOR-NEXT loop. Format 4: Exits a structural subroutine.

**EXP****EXPONENTIAL**

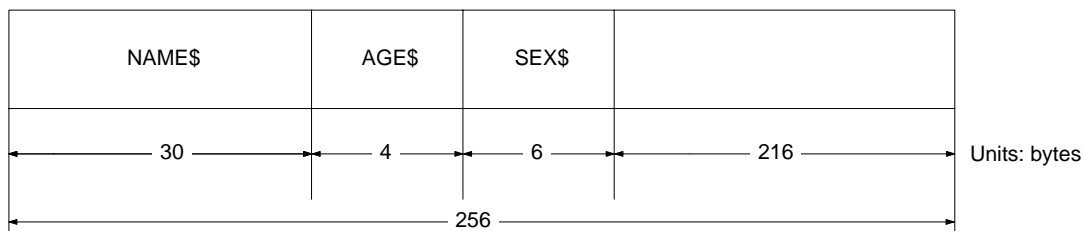
(Function)

<b>Action</b>	Determines the value of the natural number e raised to an exponential power.
<b>Format</b>	EXP ( <i>numeric expression</i> )
<b>Example</b>	A! =EXP ( 2 ) Assigns the value of e raised to the power of 2 ( $e^2$ ) to the variable A!.
<b>Description</b>	The EXP function returns the value of e ( $\approx 2.71828$ ) raised to the specified power.  The numeric expression may be in the form of an integer, long integer, or a single or double-precision real number.  The LOG function has the opposite action to the EXP function. The EXP function can be used to create other mathematical functions, such as the hyperbolic sine function (sinhX).

**FIELD #****FIELD**

(Command)

<b>Action</b>	Allocates the variable areas to the file buffer of the random access file.
<b>Format</b>	FIELD <i>file#</i> , <i>field length</i> , <i>AS character variable</i> [, <i>field length</i> , <i>AS character variable</i> ..]
<b>Example</b>	FIELD #1,30 AS NAME\$,4 AS AGE\$,6 AS SEX\$ Allocate 30 bytes for the NAME\$ variable, 4 bytes for the AGE\$ variable, and 6 bytes for the SEX\$ variable in the input buffer of the random access file opened as #1.



<b>Description</b>	To use a file buffer as the input buffer for a random access file, the areas to store character data (variable areas) must be allocated in the file buffer.  Specify the <i>file#</i> as the number in which the file was opened with the OPEN statement.  Specify the length allocated for the <i>character variable</i> with the <i>field length</i> parameter as a positive integer between 1 and 255.  Multiple character variables can be allocated. However, the total of the field lengths specified for all variables must not exceed 256 bytes. The total of the field lengths specified with a FIELD# command must equal the length of one record for random access file I/O.  The FIELD# command is used only to define the character variable names and variable areas in the file buffer. The LSET and RSET commands are used to set data in the allocated variable areas. The GET# and PUT# statements are used for data random access file I/O. It is not possible to assign data to the character variables defined with the FIELD# command with any commands except LSET and RSET. Multiple FIELD# commands can be executed to allocate different character variables to a single file buffer.
--------------------	---

## FILES

## FILES

(Command)

- Action** Displays the memory card file directory, showing the file names, sizes, and creation dates.
- Format** FILES (*drive name*)
- Example** FILES  
Displays the file directory for the memory card (drive C).
- Description** The FILES command displays the file directory for the specified *drive name*, showing the file names, sizes, and creation dates.  
The default *drive name* is drive C.  
  
The initial file creation date is 00/00/00 on 80/01/01 when the power supply is turned on. Use the DATE\$ function to apply the correct creation date to the file.

## FILTDATA

## FILTer DATA

(Command)

- Action** Specifies the line filter factors.
- Format** FILTDATA *data 0, data 1, ... , data 8* [, *global factor*]
- Example** FILTDATA -8,1,1,1,1,1,1,1,1,4  
Sets the line filter to a negative Laplacian result divided by 4.
- Description** The FILTERDATA command sets the line filter factors. It is used with the FILTER command to set the image filtering. Line filter must be selected with the FILTER function to enable the filter specified with the FILTDATA command.  
The factors are specified with the parameters *data 0* to *data 8*. The positional relationship of these pixels is shown in the diagram below.

Data 1	Data 2	Data 3
Data 4	Data 0	Data 5
Data 6	Data 7	Data 8

The nine parameters from *data 0* to *data 8* can be set to the values 0, +1, +2, +4. However, *data 0* can also be set to +8.

A filter result of 256 or higher is rounded to 255 while negative values are rounded to 0. In cases where it is desirable to restrict filtering results to 255, the filtering results can be divided by the global constant.

The global constant parameter can be set to 0, 1, 2, 4, 8, or 16. The output becomes 0 if the global constant is set to 0. The default value is 1.

## FILTER

## FILTER

(Command)

- Action** Sets the image filtering function and LUT function.
- Format** FILTER *function* [, [*LUT function*,] [, [*bias*] [, *peripheral pixels*]]]
- Example** FILTER 2,1  
Sets the sharpened image and LUT function.

**Description** The FILTER command sets the image filtering function for the camera image and frame memory image.

Set the *function* parameter to a number corresponding to one of the following filtering methods:

- 0: raw image
- 1: shading mask image
- 2: sharpened image
- 3: shading compensation image
- 4: sobel image
- 5: line filter image
- 6: (not used)
- 7: compound edge image

Set *LUT function* to 1 to enable the LUT function. The default value is 0.

The *peripheral pixels* parameter defines the output density (as 0 or 255) of the 2-pixel unstable border which arises due to filtering. The previous setting remains unchanged if the set values is not 0 or 255 or if the parameter is omitted.

The specified *bias* value is added to the image density after filtering. Specify a value between -128 and 127.

**FILTERIN**

**FILTER IN**

(Command)

**Action** Selects the input path of the image to be displayed and measured.

**Format** FILTERIN *path* [, *page#*]

**Example** FILTERIN 1

The image memory contents are output to image bus#1.

**Description** The FILTERIN command selects the camera image or an image from image memory as the image to be displayed and measured. This image is subjected to the selected filtering function.

Set the path parameter to 0 to select the camera image or 1 to select the image from the image memory. The image selected by *the path* parameter is output to image bus#1 and is also subjected to filtering before being output to image bus#0.

Omit the *page#* or set to 0.

**FIND**

**FIND**

(Command)

**Action** Searches for a specified character string and displays the line containing the character string.

**Format** Format 1: FIND *search character string* [, *line# 1*] [- [*line# 2*]] [, *A*]  
 Format 2: FIND

**Example** FIND "WHILE"  
 Displays the first line in the text program containing "WHILE".

FIND "WHILE" , 100-200 ,A  
 Displays the all lines between line 100 and line 200 in the text program containing "WHILE".

FIND  
 Continue the previous find operation.

- Description** The FIND command searches for the specified *search character string* between *line#1* and *line#2* and displays the line containing the character string. If more than one line contains the specified *search character string*, the first line found is displayed.
- Input the FIND command with no parameters to continue a previous find operation.
- If the A option is specified, each line containing the specified *search character string* is displayed.

**FIX****FIX**

(Function)

- Action** Rounds down a value to an integer.
- Format** FIX (*numeric expression*)
- Example** A=FIX(-3.3)
- Rounds down the value of the numeric expression -3.3 to an integer (-3) and assigns it to the integer variable A.
- Description** The FIX function rounds down the specified *numeric expression* and returns an integer.
- The INT and CINT functions are similar to the FIX function. Like the FIX function, the INT function also rounds down the specified value, but the INT function never returns a value larger than the value specified with the *numeric expression*. The CINT function rounds off the decimal places of the specified *numeric expression* (i.e., up or down to the nearest integer) and returns an integer. Examples of the actions of the FIX, INT, and CINT function are shown in the table below.

Expression	Specified function	Returned value
Positive numeric expression	FIX (1.7)	1
	INT (1.7)	1
	CINT (1.7)	2
Negative numeric expression	FIX (-1.7)	-1
	INT (-1.7)	-2
	CINT (-1.7)	-2

**FLASH****FLASH**

(Command)

- Action** Flashes the strobe and simultaneously reads the image to the Camera I/F Unit internal memory.
- Format** FLASH *mode* [, *period*]
- Example** FLASH 2
- The strobe flash and image input to the Camera I/F Unit internal memory are synchronized to the STEP input.
- Description** The FLASH command flashes the strobe and inputs the image to the Camera I/F Unit internal memory.
- Set the mode parameter to one of the following values:
- 0: Stop
  - 1: Operation on FLASH command only
  - 2: Operation synchronized to STEP input
  - 3: Operation each specified period

The period parameter setting is only valid when mode 3 (operation each specified period) is selected. The period is specified as a number of fields between 0 and 65535 fields. The default value is 2 fields (1 frame). Operation stops if a period of 0 is specified.

**FOR..TO..STEP-NEXT****FOR..TO..STEP-NEXT**

(Command)

<b>Action</b>	Repeats the commands between the FOR and NEXT statements.
<b>Format</b>	FOR <i>numeric variable</i> = <i>initial value</i> TO <i>final value</i> [STEP <i>increment</i> ] to NEXT [ <i>numeric variable</i> [, <i>numeric variable</i> ... ]]
<b>Example</b>	100 FOR I=1 TO 10 : : : 200 NEXT I  Repeats the statements between line 100 and line 200 ten times.
<b>Description</b>	The FOR-NEXT loop is formed by a series of statements starting with the FOR statement and ending with the NEXT statement. The series of statements inside the FOR-NEXT loop is executed the specified number of times. Normally, each FOR statement has a corresponding NEXT statement.  The specified <i>numeric variable</i> counts the number of times the FOR-NEXT loop is executed. Consequently, the same <i>numeric variable</i> name must be specified for both the FOR and NEXT statements.  The units to calculate the loop execution can be specified with STEP and an <i>increment</i> parameter. Both STEP and the <i>increment</i> parameter can be omitted. The default value of the <i>increment</i> is +1. Each time the FOR-NEXT loop is executed, the increment is added to the current value of the <i>numeric variable</i> (starting from the <i>initial value</i> the first time the loop is executed) then assigned to the <i>numeric variable</i> again.

**FRE****FREE memory**

(Function)

<b>Action</b>	Determines the amount of free space in each control area.
<b>Format</b>	FRE ( <i>function</i> )
<b>Example</b>	PRINT FRE ( 2 )  Prints the free memory size of each program area.
<b>Description</b>	The FRE function determines the free memory area in bytes of the control area specified with the function parameter.  Specify the function parameter as a value between 0 and 3, as follows: 0: Determines the free area in variable table region. 1: Determines the free area in character-string array region. 2: Determines the free area in program region. 3: Determines the total free area for functions 0 to 2.

**GATE****GATE**

(Command)

<b>Action</b>	Controls the GATE signal.
<b>Format</b>	GATE 0 or 1
<b>Example</b>	GATE 1  Turns on all GATE signals for the Terminal Block Units and Parallel I/O Units.



<b>Description</b>	The GATE command controls the GATE signal ON/OFF status of the Terminal Block Units and Parallel I/O Units. Set to 0 to turn the signal OFF or to 1 to turn the signal ON. When multiple units are connected, the GATE signals are controlled simultaneously for all units.
--------------------	---

**GCOPY****Gray COPY**

(Command)

<b>Action</b>	Copies the raw image in VRAM.
<b>Format</b>	GCOPY VRAM1, [page# 1], VRAM2, [page# 2] [, [X1] [, [Y1] [, [X2] [, [Y2] [, [X] [, [Y]]]]]]
<b>Example</b>	GCOPY 2, 0, 3, 0 Copies the entire contents of the window memory to the image memory.
<b>Description</b>	The GCOPY command copies the raw image in VRAM. Specify the copy source and destination VRAMs (VRAM1 and VRAM2). A plane VRAM cannot be specified. Use the numbers as follows: 2: Window memory 3: Image memory 4: Shading memory Omit <i>page# 1</i> and <i>page# 2</i> or set to 0. To limit the rectangular region to be copied, specify the top-left coordinates of the rectangle as X1, Y1 and the bottom-right coordinates as X2, Y2. The default values for X1, Y1 are 0, 0 and the default values for X2, Y2 are 511, 511. Specify the top-left coordinates of the copy destination rectangle as X, Y. The default values for X, Y are 0, 0. The contents of planes write protected with the MASKBIT command remain unchanged.

**GCOPY2****Gray COPY**

(Command)

<b>Action</b>	Makes enlarged and reduced copies of raw images in VRAM.
<b>Format</b>	GCOPY2 VRAM1, [page# 1], VRAM2, [page# 2] [, [XS1] [, [YS1] [, [XS2] [, [YS2] [, [XD1] [, [YD1] [, [XD2] [, [YD2]]]]]]]]]]
<b>Example</b>	GCOPY2 3, , 4, , 200, 200, 300, 300 Makes an enlarged copy of the image contained in the image memory rectangle with coordinates (200, 200), (300, 300) to the entire shading memory.
<b>Description</b>	The GCOPY2 command makes enlarged or reduced copies of the raw image in VRAM. Specify the copy source and destination VRAMs (VRAM1 and VRAM2). A plane VRAM cannot be specified. Use the numbers as follows: 2: Window memory 3: Image memory 4: Shading memory Omit <i>page# 1</i> and <i>page# 2</i> or set to 0. Specify the top-left coordinates of the copy source rectangular region as XS1, YS1 and the bottom-right coordinates as XS2, YS2. The default values for XS1, YS1 are 0, 0 and the default values for XS2, YS2 are 511, 511.

Specify the top-left coordinates of the copy destination rectangular region as *XD1*, *YD1* and the bottom-right coordinates as *XD2*, *YD2*. The default values for *XD1*, *YD1* are 0, 0 and the default values for *XD2*, *YD2* are 511, 511.

The contents of planes write protected with the MASKBIT command remain unchanged.

**GET #****GET #**

(Command)

<b>Action</b>	Reads data from a random file to the file buffer.
<b>Format</b>	GET# <i>file#</i> [, <i>numeric expression</i> ]
<b>Example</b>	GET #1, 8 Reads the #8 record from the random access file opened as file #1 to the file buffer.
<b>Description</b>	The GET# command reads data from the random access file specified with the <i>file#</i> to the file buffer. Specify the <i>file#</i> as the number in which the random access file was opened with the OPEN statement. After using a random access file, it must be closed with the CLOSE statement. Specify the number of the record to be read with the <i>numeric expression</i> . If omitted, the record after the <i>record#</i> used with the previous GET# or PUT# command is read. The data in the file buffer is assigned to the character variables allocated to variable areas with the FIELD# command. The character variables are then transferred to the program.

**GET@****GET@**

(Command)

<b>Action</b>	Reads image data from a VRAM to an array variable.
<b>Format</b>	GET@ <i>X1</i> , <i>Y1</i> , <i>X2</i> , <i>Y2</i> , <i>array name</i> , <i>VRAM</i> [, [ <i>page#</i> ] [, <i>plane#</i> ]]
<b>Example</b>	GET@ 20, 20, 200, 200, DATA1% 3, , 2 Reads image data from the rectangular region of image memory, <i>plane#</i> 2, bounded by the corner coordinates (20, 20), (200, 200) to the array variable DATA1%.
<b>Description</b>	The GET@ command reads image data from the rectangular region defined by the corner coordinates ( <i>X1</i> , <i>Y1</i> ), ( <i>X2</i> , <i>Y2</i> ) to the array variable with the name indicated by the array name. Specify the <i>VRAM</i> where the data is stored with a number, as follows: <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> Omit the <i>page#</i> or set to 0. A frame type memory is specified as the <i>VRAM</i> along with a <i>plane#</i> to store the data in the plane as binary data. The data is stored as raw image data if the <i>plane#</i> is omitted. If a plane memory is specified, the number of binary image pixels equals the number of bits in a single data array and the number of bytes in the array to store the data is given by the following equation: $((\text{horiz. pixels} + 7) \div 8) * \text{vert. pixels} + 4$

If the array is D%, for example, the qualifier is calculated as follows:

$$\text{Number bytes required} \approx 2 + 1$$

This applies when the qualifier minimum value is set to 0 with the OPTION BASE command.

If a frame memory is specified, the number of raw image pixels equals the number of bytes in a single data array and the number of bytes in the array to store the data is given by the following equation:

$$\text{horiz. pixels} * \text{vert. pixels} + 4$$

If the array is D%, for example, the number of dimensions is calculated as follows:

$$\text{Number bytes required} \approx 2 + 1$$

This applies when the qualifier minimum value is set to 0 with the OPTION BASE command.

The final 4 bytes in the equation above represents the 4 bytes at the start of array where the array width and height are stored, as follows:

D% (0) = horizontal pixels (2 bytes)

D% (1) = vertical pixels (2 bytes)

D% (2) = 1st pixels

D% (3) = 2nd pixels

## GETBLUT

## GET Binary LUT

(Command)

<b>Action</b>	Reads the contents of the present binary-coded LUT value as an array variable.
<b>Format</b>	GETBLUT <i>binary image plane#</i> , <i>array name</i> [, [ <i>qualifier</i> ] [, <i>size</i> ]]
<b>Example</b>	GETBLUT 4 , A Reads the binary LUT contents of binary image plane# 4 to array variable A.
<b>Description</b>	The GETBLUT command reads the contents of the binary LUT specified with the <i>binary image plane#</i> to the array specified with the <i>array name</i> . The qualifier specifies the first element in the data array to store the data. The default value is 0. The difference between the set qualifier value and the maximum qualifier value for the array must exceed the number of bytes set with the <i>size</i> parameter. When the binary LUT data is written to the array variable, either 256 or 512 elements are written from the start element specified by the qualifier. Other parts of the array remain unchanged.

## GETDLUT

## GET Display LUT

(Command)

<b>Action</b>	Reads the current display LUT data to an array variable.
<b>Format</b>	GETDLUT <i>region</i> , <i>array name</i> [, [ <i>qualifier</i> ]]
<b>Example</b>	GETDLUT 1 , A Reads the display LUT data from inside the window to array variable A.
<b>Description</b>	The GETDLUT command reads the current display LUT data to the array specified with the <i>array name</i> . Individual display LUTs are available for inside and outside the window. Specify which LUT is required with the <i>region</i> parameter. Set region to 0 to read the LUT data from outside the window or 1 to read the data from inside the window. The <i>qualifier</i> specifies the first element in the data array to store the LUT data. The default value is 0.

**GETDLVL****GET Display LeVeL**

(Command)

<b>Action</b>	Determines the display level of the image.
<b>Format</b>	GETDLVL ( <i>image type</i> )
<b>Example</b>	GETDLVL ( 3 ) Determines the displayed brightness of white in the binary image.
<b>Description</b>	The GETDLVL function determines the displayed brightness of the image specified by the <i>image type</i> parameter, as follows: <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Mask image</li> <li>3: Binary image, white</li> <li>4: Binary image, black</li> <li>5: Window memory increments</li> <li>6: Paint/pattern matching window memory increments</li> </ul>

**GETLUT****GET LUT**

(Command)

<b>Action</b>	Reads the current filtering LUT data to an array variable.
<b>Format</b>	GETLUT <i>array name</i> [, [ <i>qualifier</i> ] [, <i>size</i> ]]
<b>Example</b>	GETLUT A Reads the filtering LUT data to array variable A.
<b>Description</b>	The GETLUT command reads the current filtering LUT data to the array specified with the <i>array name</i> . The <i>qualifier</i> specifies the first element in the data array to store the LUT data. The default value is 0. The <i>size</i> parameter specifies the number of data bytes to be stored. Set the size to 0 to store 256 bytes or to any other value to store 512 bytes. The default value is 0. The difference between the set <i>qualifier</i> value and the maximum <i>qualifier</i> value for the array must exceed the number of bytes set with the <i>size</i> parameter. When the filtering LUT data is written to the array variable, either 256 or 512 elements are written from the start element specified by the <i>qualifier</i> . Other parts of the array remain unchanged.

**GOSUB****GOSUB**

(Command)

<b>Action</b>	Branches to a specified subroutine.
<b>Format</b>	GOSUB <i>line# or label</i>
<b>Example</b>	GOSUB *START Branches to the subroutine starting from the line labelled *START.
<b>Description</b>	The GOSUB command branches control to the subroutine starting with the specified <i>line number or label</i> . Control returns to the GOSUB statement position when the RETURN statement at the end of the subroutine is executed. Specify the first line of the subroutine with the <i>line# or label name</i> as the GOSUB command parameter.

When subroutines are nested, GOSUB and RETURN statements must always be used in pairs.

**GOTO**

**GO TO**

(Command)

<b>Action</b>	Unconditionally jumps to a specified line.
<b>Format</b>	FORMAT1: GO TO <i>line# or label</i> FORMAT2: GOTO <i>line# or label</i>
<b>Example</b>	GOTO *START Program operation jumps to the line labelled *START.
<b>Description</b>	The GOTO command unconditionally jumps control to the line with the specified line number or label. The action of the GOTO and GO TO statements is identical.

**HELP**

**HELP**

(Command)

<b>Action</b>	Displays help messages.
<b>Format</b>	HELP [ <i>"command or function name"</i> ]
<b>Example</b>	HELP "A*" Displays help messages for the commands and functions beginning with "A".
<b>Description</b>	The HELP command displays help messages for the commands and functions specified with the <i>command name</i> parameter. The wildcard characters (?, *) can be used in the <i>command name</i> . A list of command and function names is displayed if the <i>command name</i> parameter is omitted. No message is displayed if an unregistered command or function name is specified with the <i>command name</i> .

**HELP ON/OFF/STOP**

**HELP key ON/OFF/STOP**

(Command)

<b>Action</b>	Disables, enables, or stops interrupts from the HELP Key.
<b>Format</b>	HELP ON <i>or</i> OFF <i>or</i> STOP
<b>Example</b>	HELP ON Operation branches to an interrupt processing routine when the HELP Key is pressed.

<b>Description</b>	The HELP command controls branching to an interrupt processing routine when the HELP Key is pressed.
HELP ON:	The HELP ON statement enables the interrupt processing routine when the HELP Key is pressed. When the HELP Key is pressed, operation branches to the interrupt processing routine at the line# or label defined with the ON HELP GOSUB statement.
HELP OFF:	The HELP OFF statement disables the interrupt processing routine when the HELP Key is pressed. When the HELP Key is pressed, operation does not branch to an interrupt processing routine.
HELP STOP:	The HELP STOP statement stops interrupt processing when the HELP Key is pressed. When the HELP Key is pressed, operation does not immediately branch to the interrupt processing routine. However, immediately branching is enabled by the HELP ON statement, operation branches to the interrupt processing routine at the line# or label defined with the ON HELP GOSUB statement.

**HEX\$**

**HEX\$**

(Function)

<b>Action</b>	Converts a numeric expression to a hexadecimal character string.
<b>Format</b>	HEX\$ ( <i>numeric expression</i> )
<b>Example</b>	A\$=" &H"+HEX\$( 100) Converts the decimal value 100 to a hexadecimal character string, concatenates this character string with the character string " &H", and assigns the result to the character variable A\$. This equation is identical to A\$=" &H64".
<b>Description</b>	The HEX\$ function converts a decimal value to a hexadecimal character string. It does not add the " &H" prefix to indicate a hexadecimal character string. Specify the <i>numeric expression</i> as a decimal numeric constant or numeric variable between $-2^{31}$ and $2^{31}-1$ . Any decimal places in the specified <i>numeric expression</i> are rounded off to create an integer which is converted to the hexadecimal character string. The relationship between the decimal <i>numeric expression</i> and hexadecimal character string is as follows:

Numeric expression (decimal)	Hexadecimal character string
-2147483648 to -1	"80000000" to "FFFFFFF"
0 to 2147483647	"0" to "7FFFFFFF"

To convert a hexadecimal character string back to numeric data, append the " &H" prefix to indicate a hexadecimal character string, then convert the character string with the VAL function.

**HISTGRAM**

**HISToGRAM**

(Command)

<b>Action</b>	Reads the density histogram from the image memory.
<b>Format</b>	HISTGRAM [ <i>page#</i> ], X1, Y1, X2, Y2, <i>array name</i> [, <i>qualifier</i> ]
<b>Example</b>	HISTGRAM , 0, 0, 100, 100, H Stores the density histogram contained in the image memory rectangle with corner coordinates (0, 0), (100, 100) to array name H.
<b>Description</b>	The HISTGRAM command reads the density histogram from the image in the image memory. Omit the <i>page#</i> or set to 0.

The density histogram is read from the rectangular region with the specified corner coordinates *X1*, *Y1* and *X2*, *Y2*.

Specify the name of the array to store the density histogram with the *array name* parameter.

The *qualifier* specifies the first element in the data array to store the density histogram. The default value is 0.

The array must have at least 256 elements between the specified *qualifier* value and the maximum qualifier value.

**IF . . GOTO-ELSE**

**IF . . GOTO-ELSE**

(Command)

<b>Action</b>	Controls the program flow with a specified condition.
<b>Format</b>	<i>IF conditional expression GOTO line # or label ELSE statement or line # or label</i>
<b>Example</b>	<pre>IF A=1 GOTO 100 ELSE 200</pre> Branches to line 100 if variable A equals 1 and branches to line 200 if variable A is not equal to 1.
<b>Description</b>	The IF ... GOTO-ELSE command control the program flow with the specified <i>conditional expression</i> .  If the conditional expression is true (not 0), program operation jumps to the <i>line#</i> or <i>label</i> specified after GOTO. The statement, <i>line#</i> or <i>label</i> specified after ELSE is ignored.  If the <i>conditional expression</i> is false (0), program operation jumps to the <i>line#</i> or <i>label</i> specified after ELSE or executes the statement specified after ELSE. The <i>line#</i> or <i>label</i> specified after GOTO is ignored.  The ELSE statement can be omitted.

**IF . . THEN-ELSE**

**IF . . THEN-ELSE**

(Command)

<b>Action</b>	Controls the program flow with a specified condition.
<b>Format</b>	<i>IF conditional expression THEN statement or line # or label ELSE statement or line # or label</i>
<b>Example</b>	<pre>IF A\$="y" THEN GOSUB *START ELSE B=B+1</pre> Branches to the subroutine labelled *START if variable A\$ equals "y", otherwise increments variable B by 1.
<b>Description</b>	The IF ... THEN-ELSE command control the program flow with the specified <i>conditional expression</i> .  If the <i>conditional expression</i> is true (not 0), program operation jumps to the <i>line#</i> or <i>label</i> specified after THEN or executes the statement specified after THEN. The <i>statement</i> , <i>line#</i> or <i>label</i> specified after ELSE is ignored.  If the <i>conditional expression</i> is false (0), program operation jumps to the <i>line#</i> or <i>label</i> specified after ELSE or executes the statement specified after ELSE. The <i>statement</i> , <i>line#</i> or <i>label</i> specified after THEN is ignored.  The ELSE statement can be omitted.

## IF...THEN-ELSEIF-ELSE-END IF

(Command)

**Action** Evaluates specified conditions.

**Format** IF *conditional expression* THEN  
*Statement in THEN block*  
ELSEIF *conditional expression* THEN  
*Statement in ELSE IF block*  
⋮  
[ELSE  
*Statement in ELSE block*]  
END IF

**Example**

```
IF ERR=0 THEN
    PRINT "NORMAL"
ELSEIF ERR=2 THEN
    PRINT "SYNTAX ERROR"
ELSEIF ERR=13 THEN
    PRINT "TYPE MISMATCH"
ELSE
    PRINT "OTHER ERROR"
END IF
```

Defines different actions when the variable ERR equals 0, 2, 13, or some other value.

**Description** The IF ... THEN-ELSEIF-ELSE-END IF commands evaluate specified *conditional expressions*.

Executes the subsequent Statement in THEN block if the specified *conditional expression* is true (not 0). Jumps to the next ELSE IF, ELSE, or END statement if the *conditional expression* is false (0).

Multiple ELSE IF statements may be used or they may be omitted.

The ELSE statement may be omitted.

The END IF statement is required. It cannot be omitted.

The GOTO command must not be used to jump into or out of a IF ... THEN-ELSEIF-ELSE-END IF statement block.

## IMGLOAD

## IMaGe LOAD

(Command)

**Action** Loads image data to VRAM.

**Format** IMGLOAD *file name*, X, Y, VRAM, [, [*page#*] [, *plane#*]]

**Example** IMGLOAD "IMG1" , 0 , 0 , 2 , , 5

Writes the file IMG1 from the memory card to the rectangular region with top-left coordinates (0, 0) in window memory plane 5.

**Description** The IMGLOAD command loads image data saved with the IMGSAVE command to VRAM.

Specify the image data with the *file name* parameter.

The data is saved to rectangular region. Specify the top-left coordinates of this rectangular region with the X, Y parameters.



Specify the VRAM where the data is loaded with a number, as follows:

- 0: Character memory
- 1: Graphic memory
- 2: Window memory
- 3: Image memory
- 4: Shading memory

Omit the *page#* or set to 0.

Specify the *plane#* when loading binary image data to a frame VRAM. An error occurs if an attempt is made to write raw image data to a plane VRAM. If image data is written to a frame VRAM, it cannot be loaded to planes write protected with the MASKBIT command.

## IMGSAVE

## IMaGe SAVE

(Command)

<b>Action</b>	Saves image data from VRAM.
<b>Format</b>	IMGSAVE <i>file name</i> X1, Y1, X2, Y2, VRAM, [, [ <i>page#</i> ] [, <i>plane#</i> ]] [, <i>compression function</i> ]]]
<b>Example</b>	<p>IMGSAVE "IMG1" , 10 , 10 , 100 , 100 , 2 , , 3</p> <p>Saves the rectangular region with corner coordinates (10, 10), (100, 100) from the window memory plane 3 to the file IMG1 in the memory card with no compression.</p>
<b>Description</b>	<p>The IMGSAVE command saves image data from VRAM.</p> <p>The contents of the image memory are saved to a file in the memory card with the specified <i>file name</i>. Only the image inside the rectangular region with corner coordinates (X1, Y1), (X2, Y2) is saved.</p> <p>Specify the <i>VRAM</i> from which the data is saved with a number, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the <i>plane#</i> when saving binary image data or window data from a single plane of a frame VRAM. The data is treated as raw image data if the <i>plane#</i> is omitted.</p> <p>Specify the <i>compression function</i>, as follows:</p> <ul style="list-style-type: none"> <li>0: When saving binary image data (No compression)</li> <li>1: When saving binary data (Compression using run-length method)</li> <li>2: When saving shading data and raw image data (High compression)</li> </ul> <p>The default value is 0. When the value 2 is specified, the rectangular region must be set to (0,0), (511,511).</p>

## INKEY\$

## INput KEY\$

(Function)

<b>Action</b>	Determines the character input from the keyboard.
<b>Format</b>	INKEY\$
<b>Example</b>	<p>A\$=INKEY\$</p> <p>Assigns the key character of the key pressed on the keyboard to variable A\$.</p>

<b>Description</b>	<p>The INKEY\$ function returns the key character of the key pressed when the function is executed. The function returns a null string (" ") if no key is pressed (that is, the keyboard buffer is empty). Refer to the Note below.</p> <p>Data input with the INKEY\$ function is not displayed on the screen. Unlike other input commands, operation does not wait for a key to be pressed after the function is executed.</p>
<b>Note</b>	<p>Information on the pressed keys is stored in the keyboard buffer even if no key input statement is executed. If data remains in the keyboard buffer when the INKEY\$ function is executed, the function returns the first key character from the keyboard buffer, regardless of the key pressed at that time.</p>

**INPUT****INPUT**

(Command)

<b>Action</b>	Assigns data input from the keyboard to a variable.
<b>Format</b>	INPUT [ <i>prompt character string</i> { , or ;}] <i>variable</i> [, <i>variable</i> ...]
<b>Example</b>	<pre>INPUT "NAME" ; B\$</pre> <p>Displays "NAME" on the screen and waits for keyboard input of the character data to be assigned to the character variable B\$.</p>
<b>Description</b>	<p>Operation waits for data input from the keyboard when the INPUT command is executed. Enter data delimited by commas (,) for the specified number of variables and press the Return Key to assign the data to the variables.</p> <p>Specify a character string for the <i>prompt character string</i>. This character string is displayed to prompt for input of the required data.</p> <p>The <i>prompt character string</i> must be separated from the subsequent <i>variable</i> by a comma (,) or semicolon (;). The specified <i>prompt character string</i> only is displayed if a comma is used. The specified <i>prompt character string</i> followed by a question mark (?) and a single space is displayed if a semicolon is used. If no specified <i>prompt character string</i> is specified, a question mark (?) and a single space are displayed.</p> <p>Multiple variables can be specified delimited by commas (,). The variables can be either numeric or character variables.</p> <p>If the Return Key is pressed without entering values, zero (0) is assigned to numeric variables and a null string (" ") is assigned to character variables.</p>

**INPUT#****INPUT#**

(Command)

<b>Action</b>	Reads data from a sequential access file and assigns it to variables.
<b>Format</b>	INPUT # <i>file#</i> , <i>variable</i> [, <i>variable</i> ...]
<b>Example</b>	<pre>INPUT #1, A\$, B, C</pre> <p>Reads three data items (character, numeric, numeric) sequentially from file #1 and assigns the data to variable A\$ (character variable), B (numeric variable), and C (numeric variable).</p>
<b>Description</b>	<p>Specify the <i>file#</i> as the number in which the random access file was opened with the OPEN statement.</p> <p>The <i>variables</i> can be specified either as numeric or character variables. The specified variable format must match the corresponding data format.</p> <p>The data is read from the file as numeric or character variables, complying with the rules governing these data types. Unlike the INPUT command, the INPUT# command displays no prompt message or question mark (?).</p>

**Numeric Variables:** Leading spaces are ignored and the data is read in from the first non-blank character. The data up to the next blank, comma (,), or linefeed character (&HA) is assigned to the numeric variable.

**Character Variables:** Leading spaces are ignored and the data is read in from the first non-blank character. The data up to the next blank, comma (,), or linefeed character (&HA) is assigned to the character variable.

If the first character is a double-quotation mark ("), subsequent data is considered to be contained in double-quotation marks (") and all characters and blanks up to the next double-quotation mark (") are assigned to the character variable. Therefore, a comma (,) or linefeed character (&HA) must be enclosed in double-quotation marks (") to be assigned to a variable.

**INPUT\$****INPUT\$**

(Function)

<b>Action</b>	Reads a specified length of data from a sequential access file, the RS-232C port, or the keyboard.
<b>Format</b>	INPUT\$ ( <i>character string length</i> [, <i>file#</i> ])
<b>Example</b>	A\$=INPUT\$ ( 5 , #3 ) Reads 5 characters from file #3 and assigns them to variable A\$.
<b>Description</b>	Specify the <i>file#</i> to read the data from with the <i>file#</i> parameter. Data is read from the keyboard if the <i>file#</i> parameter is omitted. Unlike data input with the INPUT or LINE INPUT command, data input from the keyboard using the INPUT\$ function is not displayed on the screen. Specify the <i>character string length</i> as a positive integer value. The program operation waits for input of the specified number of characters. If the length of the data input from the RS-232C port exceeds the number of characters specified with the <i>character string length</i> , the remaining data is read to the next INPUT\$ function.

**INPUT WAIT****INPUT WAIT**

(Command)

<b>Action</b>	Inputs data from the keyboard with a time limitation.
<b>Format</b>	INPUT WAIT <i>wait time</i> , [ <i>prompt character string</i> { , <b>or</b> ;}] <i>variable</i> [, <i>variable..</i> ]
<b>Example</b>	INPUT WAIT 150 , "ANSWER" ; AN\$ Displays the input prompt message "ANSWER" on the screen and waits 15 seconds for keyboard input of the character data to be assigned to the character variable A\$.
<b>Description</b>	Operation waits for the specified <i>wait time</i> for data input from the keyboard when the INPUT WAIT command is executed. Enter data delimited by commas (,) for the specified number of variables and press the Return Key within the specified <i>wait time</i> to assign the data to the <i>variables</i> . The INPUT WAIT command is identical to the INPUT command, except for the time limit. If the Return Key is pressed without entering values for the <i>variables</i> , zero (0) is assigned to numeric variables and a null string (" ") is assigned to character variables. If the time specified by <i>wait time</i> passes, the variables will not change. Specify the <i>wait time</i> in units of 0.1 second.

**INSTR****IN STRing**

(Function)

<b>Action</b>	Determines the position of the specified characters in the character string
<b>Format</b>	INSTR ([ <i>start position</i> ,] <i>character string 1</i> , <i>character string 2</i> )
<b>Example</b>	<p>X=INSTR( , A\$, "XYZ" )</p> <p>Determines the position of character string XYZ from the start of the character string A\$ and assigns the position as a number of bytes to variable X.</p>
<b>Description</b>	<p>The INSTR function finds a specified <i>character string 2</i> in a <i>character string 1</i> of one-byte characters and returns the position of <i>character string 2</i> in bytes from the start of <i>character string 1</i>.</p> <p>The function returns the value 0 if the specified character string 2 does not exist in character string 1. The INSTR function still returns the number of 1-byte characters if it is used to find a character string in a string of 2-byte characters. In this case, use the KINSTR function instead.</p> <p>Specify a value with the <i>start position</i> parameter to set the the position to start searching <i>character string 1</i>. Specify the value between 1 and the number of characters in <i>character string 1</i>. If omitted, searching starts from the beginning of the character string. The function returns the value 0 if the specified <i>start position</i> is larger than the number of characters in <i>character string 1</i>.</p> <p>Specify a character constant or a character variable for <i>character string 1</i>, the character string to search. The function returns the value 0 if the specified <i>character string 1</i> is a null string (" ").</p> <p>Specify a character constant or a character variable for <i>character string 2</i>, the character string to be searched for. The function returns the specified <i>start position</i> value if <i>character string 2</i> is specified as a null string (" ").</p>

**INT****INTEger**

(Function)

<b>Action</b>	Rounds down a value to an integer.
<b>Format</b>	INT ( <i>numeric expression</i> )
<b>Example</b>	<p>A=INT( 3.3 )</p> <p>Rounds down the value of the numeric expression 3.3 to an integer 3 and assigns it to the integer variable A.</p>
<b>Description</b>	<p>The INT function rounds down the specified <i>numeric expression</i> and returns an integer not exceeding the value specified with the <i>numeric expression</i>.</p> <p>The FIX and CINT functions are similar to the INT function. However, the FIX function simply cuts off (rounds down) the decimal places from the specified value. The CINT function rounds off the decimal places of the specified <i>numeric expression</i> (i.e., up or down to the nearest integer) and returns an integer. Examples of the actions of the FIX, INT, and CINT function are shown in the table below.</p>

Expression	Specified function	Returned value
Positive numeric expression	FIX (1.7)	1
	INT (1.7)	1
	CINT (1.7)	2
Negative numeric expression	FIX (-1.7)	-1
	INT (-1.7)	-2
	CINT (-1.7)	-2

**INTR ON/OFF/STOP****INTeRrupt ON/OFF/STOP**

(Command)

<b>Action</b>	Disables, enables, or stops interrupts with the STEP signal.
<b>Format</b>	INTR ON <i>or</i> OFF <i>or</i> STOP
<b>Example</b>	INTR ON Enables STEP interrupts.
<b>Description</b>	<p>The INTR command control execution of the interrupt subroutine defined with the ON INTR GOSUB command when a STEP input is generated from a Terminal Block Unit or Parallel I/O Unit. The rising edge of the STEP input signal is detected.</p> <p>The INTR ON statement enables the STEP interrupt processing. When the STEP signal is input, the interrupt routine is executed immediately.</p> <p>The INTR OFF statement disables the STEP interrupt processing. The STEP input is ignored.</p> <p>The INTR STOP statement stops STEP interrupt processing. An input STEP signal is stored in memory, and when STEP interrupt processing is enabled again with the INTR ON command, operation immediately branches to the STEP interrupt processing routine. Operation jumps to the STEP interrupt processing routine only once, regardless of the number of times the STEP signal was input.</p>

**IPL****Initial Program Loading**

(Command)

<b>Action</b>	Sets the OVL boot-up mode.
<b>Format</b>	IPL [[0 <i>or</i> 1 <i>or</i> file name], [numeric value 1], [numeric value 2], [numeric value 3], [numeric value 4], [numeric value 5]]
<b>Example</b>	<p>IPL 1 Automatic boot-up of the program in the text region when the power is turned on.</p> <p>IPL "C:FILE" Loads and executes "C:FILE" when the power is turned on.</p> <p>IPL Displays the current OVL boot-up mode.</p>
<b>Description</b>	<p>Set the IPL command initial parameter to 1 to automatically load and boot-up the program in the text region. Set the parameter to 0 to disable automatic boot-up. Set a <i>file name</i> as the parameter to load and execute a saved file.</p> <p>The <i>numeric value 1</i> parameter specifies the number of files which can be open simultaneously. Set a value between 1 and 11.</p> <p>The <i>numeric value 2</i> parameter specifies the size of the array variable or character variable region in Kbytes. Set a value between 1 and 62 Kbyte.</p> <p>The <i>numeric value 3</i> parameter specifies the size of the user stack region in Kbytes. Set a value between 1 and 2 Kbyte.</p> <p>The <i>numeric value 4</i> parameter specifies the size of the compile region in Kbytes. Set a value between 2 and 8 Kbyte.</p> <p>The <i>numeric value 5</i> parameter specifies the number of lines displayed on the screen. Set the value to 20 or 25.</p> <p>If all parameters are omitted, the current OVL boot-up mode status is displayed.</p>

**JIS\$****JIS\$**

(Function)

<b>Action</b>	Determines Shift JIS code for a 2-byte character.
<b>Format</b>	JIS\$ ( <i>character string</i> )
<b>Example</b>	J\$=JIS\$ ("ABC") Assigns the Shift JIS code (8260) corresponding to the first character of the character string "ABC".
<b>Description</b>	The JIS\$ function returns the hexadecimal character for the first 2-byte character in the specified <i>character string</i> . If the specified character string contains 1-byte characters, the code is returned for the first and second bytes of the string. An error occurs if a null string (" ") or a string containing one byte is specified. The KNJ\$ function has the opposite function to the JIS\$ function. The KNJ\$ function returns the 2-byte character corresponding to the specified 4-digit hexadecimal Shift JIS code. The F300 uses the Shift JIS codes.

**KACNV\$****Kanji AnK CoNVert\$**

(Function)

<b>Action</b>	Converts 2-byte characters in a character string to 1-byte characters.
<b>Format</b>	KACNV\$ ( <i>character string</i> )
<b>Example</b>	C\$=KACNV\$ ("OMRON F300") Converts the 2-byte characters "OMRON" in the character string "OMRON F300" to the 1-byte characters "OMRON F300" and assigns the entire character string to variable C\$.
<b>Description</b>	The KACNV\$ function converts an alphabetic, numeric, or kana 2-byte character to a 1-byte character of identical meaning. The KACNV\$ function returns a character string containing only 1-byte characters. Specify the character string with a character constant or character variable as a character string containing alphabetic, numeric, and kana characters defined with both 1-byte and 2-byte character codes. An error occurs if the character string contains a Kanji character or a 2-byte character for which no 1-byte character is defined. The AKCNV\$ function has the opposite action to the KACNV\$ function. The AKCNV\$ function converts 1-byte alphabetic, numeric, and kana characters in a character string to equivalent 2-byte characters.

**KEXT\$****Kanji EXtract \$**

(Function)

<b>Action</b>	Extracts either 1-byte or 2-byte characters from the character string.
<b>Format</b>	KEXT\$ ( <i>character string, function</i> )
<b>Example</b>	K\$=KEXT\$ ("OMRON VISION") Assigns only the 1-byte characters to variable K\$. In this case, K\$ contains the string "VISION".
<b>Description</b>	The KEXT\$ function extracts either the 1-byte or 2-byte characters from a character string containing by character types. Specify the <i>character string</i> from which the characters are selected as a character constant or a character variable.

Specify the *function* parameter as either 0 or 1. The meanings of these settings are described in the table.

Function	Description
0	Extract the 1-byte characters
1	Extract the 2-byte characters

The KEXT\$ function returns a null string (" ") if the type of character specified with the function parameter does not exist in the character string.

**KEY****KEY**

(Command)

<b>Action</b>	Assigns any character string to the function keys.
<b>Format</b>	KEY [key#, character expression]
<b>Example</b>	KEY 1, "TOTAL" Sets the character string "TOTAL" to function key 1 and displays this character string on in the input guide.
<b>Description</b>	The KEY command sets the display for the function key (F1 to F10) with the specified key# using a character string or control code. Specify the key# parameter as a value between 1 and 10. This number corresponds to one of the functions keys (F1 to F10). Specify the character expression with up to 15 1-byte characters and control codes (CHR\$(1) to CHR\$(31) and CHR\$(127)). The control codes cannot be input from the keyboard. Specify the codes using the CHR\$ function with character codes linked by "+" signs. Each control code occupies one byte. The input guide displays the first five standard-sized characters. If the key# and character expression parameters are omitted, the settings revert to the settings when the OVL was booted up.

**KEY LIST****KEY LIST**

(Command)

<b>Action</b>	Displays the function key settings on the screen.
<b>Format</b>	KEY LIST
<b>Example</b>	KEY LIST Displays the settings of all the function keys on the screen.
<b>Description</b>	The KEY LIST command displays the present settings of the ten function keys.

**KEY ON/OFF/STOP****KEY ON/OFF/STOP**

(Command)

<b>Action</b>	Disables, enables, or stops interrupts from the function keys.
<b>Format</b>	KEY [(key#)] ON <b>or</b> OFF <b>or</b> STOP
<b>Example</b>	KEY (10) ON Operation branches to an interrupt processing routine when the function key# 10 (F10) is pressed.
<b>Description</b>	The KEY command controls branching to an interrupt processing routine when a function key is pressed. Specify the key# parameter as a value between 1 and 10 corresponding to a function key number. The KEY ON/OFF/STOP command applies to all keys if the key# is not specified.

KEY ON:	The KEY ON statement enables the interrupt processing routine when the function key is pressed. When the function key is pressed, operation branches to the interrupt processing routine at the line# or label defined with the ON KEY GOSUB statement.
KEY OFF:	The KEY OFF statement disables the interrupt processing routine when the function key is pressed. When the function key is pressed, operation does not branch to an interrupt processing routine.
KEY STOP:	The KEY STOP statement stops interrupt processing when the function key is pressed. When the function key is pressed, operation does not immediately branch to the interrupt processing routine but the pressed status is stored in memory. Immediately branching is enabled by the KEY ON statement, operation branches to the interrupt processing routine at the line# or label defined with the ON KEY GOSUB statement.

**Note** The interrupt subroutine must be defined with the ON KEY GOSUB statement before a KEY ON/OFF/STOP command is executed.

**KEYIN****KEYIN**

(Function)

<b>Action</b>	Reads the input status of the console keys.								
<b>Format</b>	KEYIN ( <i>input mode</i> )								
<b>Example</b>	A=KEYIN( 1 ) Waits for a key input from the console and assigns the input status to variable A.								
<b>Description</b>	Returns the input status of the console keys. If the <i>input mode</i> parameter is set to 0, the function returns 0 if no key is pressed at the time the function is executed. If the <i>input mode</i> parameter is set to 1, operation waits until a key is pressed. The function returns a value between 0 and 255. Each bit of the binary representation of the returned value corresponds to a pressed key, as follows: <table style="margin-left: 40px;"> <tr> <td>Bit 0: j</td> <td>Bit 4: ENT</td> </tr> <tr> <td>Bit 1: Q</td> <td>Bit 5: ESC</td> </tr> <tr> <td>Bit 2: A</td> <td>Bit 6: HELP</td> </tr> <tr> <td>Bit 3: i</td> <td>Bit 7: SHIFT (see note)</td> </tr> </table>	Bit 0: j	Bit 4: ENT	Bit 1: Q	Bit 5: ESC	Bit 2: A	Bit 6: HELP	Bit 3: i	Bit 7: SHIFT (see note)
Bit 0: j	Bit 4: ENT								
Bit 1: Q	Bit 5: ESC								
Bit 2: A	Bit 6: HELP								
Bit 3: i	Bit 7: SHIFT (see note)								

**Note** The status of the SHIFT Key is returned only when it is pressed simultaneously with another key.

**KILL****KILL**

(Command)

<b>Action</b>	Deletes the file with the specified filename from the memory card.
<b>Format</b>	KILL <i>filename</i>
<b>Example</b>	KILL "BASIC.DAT" Delete the file BASIC.DAT from the memory card.
<b>Description</b>	Specify the name of a file existing in the memory card as a character string with the filename parameter. An error occurs if the specified filename does not exist in the memory card. The filename parameter can specify any program or data file in the memory card. However, an error occurs if the KILL command is executed on a write-protected file.



An error occurs if the KILL command is executed on a file opened with the OPEN statement. Close the file with the CLOSE statement before deleting it with the KILL command. A file cannot be deleted with the KILL command if the file attribute is set to write-protected with the SET command. Use the SET command to remove the write protection before deleting the file with the KILL command.

**KINPUT****Kanji INPUT**

(Command)

<b>Action</b>	Automatically sets the Japanese input mode and reads data from the keyboard.
<b>Format</b>	KINPUT <i>character variable</i>
<b>Example</b>	KINPUT KANJI\$ Waits for keyboard input of the Japanese character data to assign to the character variable KANJI\$.
<b>Description</b>	The Japanese input mode is selected automatically when the KINPUT command is executed and the system waits for input of a Japanese (2-byte) character from the keyboard. After the data is input, it is assigned to the <i>character variable</i> when the Return Key is pressed.

**KINSTR****Kanji IN STRing**

(Function)

<b>Action</b>	Determines the position of the specified characters in the character string
<b>Format</b>	KINSTR ([ <i>start position</i> ,] <i>character string 1</i> , <i>character string 2</i> )
<b>Example</b>	K=KINSTR (2, A\$, JAPAN) Searches for the character string "JAPAN" from the second character of the character string A\$ and assigns the position from the start of the array as a number of bytes to variable K.
<b>Description</b>	The KINSTR function finds a specified <i>character string 2</i> in a <i>character string 1</i> and returns the position of <i>character string 2</i> from the start of <i>character string 1</i> . 1-byte and 2-byte characters are each counted as a single character. Specify a value with the <i>start position</i> parameter to set the the position to start searching <i>character string 1</i> . Specify the value between 1 and the number of characters in <i>character string 1</i> . If omitted, searching starts from the beginning of the character string. The function returns the value 0 if the specified <i>start position</i> is larger than the number of characters in <i>character string 1</i> . Specify a character constant or a character variable for <i>character string 1</i> , the character string to search. The function returns the value 0 if the specified <i>character string 1</i> is a null string (""). Specify a character constant or a character variable for <i>character string 2</i> , the character string to be searched for. The function returns the specified <i>start position</i> value if <i>character string 2</i> is specified as a null string (""). The function returns the value 0 if the specified <i>character string 2</i> does not exist in <i>character string 1</i> .

**KLEN****Kanji LENgth**

(Function)

<b>Action</b>	Determines the number of characters in a character string including 2-byte characters.
<b>Format</b>	KLEN ( <i>character string</i> [, <i>function</i> ])
<b>Example</b>	L=KLEN ("BASIC") Assigns the length of the character string "BASIC" (5) to variable L.

- Description** The KLEN function returns the length of a specified character string. 1-byte and 2-byte characters are each counted as a single character.
- Specify a character constant or a character variable for character string 1, the character string to search. The character string can contain both 1-byte and 2-byte characters.
- Specify the function parameter as an integer. The default value is 0. The meaning of the function parameter is shown below.

Function	Description
0	Determines the total number of 1-byte and 2-byte characters in the character string.
1	Determines the total number of 1-byte characters only.
2	Determines the total number of 2-byte characters only.
3	Determines the total number of wide characters only.

**KMID\$****Kanji MIDDLE \$**

(Function)

- Action** Extracts part of a character string containing 2-byte Japanese characters.
- Format** KMID\$ (*character variable*, *start position* [, *number of characters*])
- Example** KA\$=KMID\$(K\$,1,10)  
Extracts 10 characters from the first character of the character string K\$ and assigns them to the character string KA\$.
- Description** Extracts a specified length of character string from a character string containing 2-byte Japanese characters.
- Specify a character constant or a character variable for *character variable* to search. Do not specify a null string ("").
- Specify the number of characters to be extracted with the *number of characters* parameter. The function returns a null string ("") if the specified number of characters is negative or exceeds the actual number of characters between the *start position* and the right end of the character variable. Set the *start position* to 1 to select a string from the start of the specified *character variable*.
- Specify the number of characters to be extracted with the *number of characters* parameter. All characters to the right of the *start position* are replaced if the *number of characters* is omitted or if the *number of characters* exceeds the actual number of characters between the *start position* and the right end of the *character variable*.

**KNJ\$****KaNJi\$**

(Function)

- Action** Determines the character corresponding to a Shift JIS character code.
- Format** KNJ\$ (*character string*)
- Example** K\$=KNJ\$("8260")  
Assigns 2-byte character "A" corresponding to the Shift JIS code 8260 to variable K\$.
- Description** The KNJ\$ function returns the 2-byte character corresponding to the specified 4-digit hexadecimal Shift JIS code.
- Specify a Shift JIS code with the character string.

The JIS\$ function has the opposite function to the KNJ\$ function. The JIS\$ function returns a 4-digit hexadecimal Shift JIS code corresponding to the first 2-byte character in a character string.

**KPLOAD****Kanji Pattern LOAD**

(Command)

<b>Action</b>	Registers user-defined character patterns in the F300.
<b>Format</b>	KPLOAD <i>character code, integer array name</i>
<b>Example</b>	<p>KPLOAD &amp;HEC40 , CHRPTN%</p> <p>Register the Kanji pattern defined in array CHRPTN% as the Kanji code &amp;HEC40.</p>
<b>Description</b>	<p>The KPLOAD command registers user-defined character patterns in the F300.</p> <p>Specify the code to be registered with the character code parameter. Use the codes between &amp;HF8 and &amp;HFF for standard characters and the codes between &amp;HEB9F and &amp;HEBFC or between &amp;HEC40 and &amp;HEC61 for wide characters. An error occurs if the character code is specified outside these ranges.</p> <p>Specify the array variable containing the character pattern with the integer array name. The array variable must previously be declared with the DIM command as a one-dimensional array with 16 elements.</p> <p>Store the dot image of the character pattern in elements 1 to 16 of the array variable in the format shown below:</p> <pre> Array element 1 = &amp;H1281 Array element 2 = &amp;H2242 . . . Array element 16 = &amp;H0 </pre>

**KPOS****Kanji POSition**

(Function)

<b>Action</b>	Determines the number of bytes to the specified character position in the character string which includes 2-byte Japanese characters.
<b>Format</b>	KPOS ( <i>character string, character position</i> )
<b>Example</b>	<p>K = KPOS ( "ABCDEF" , 5 )</p> <p>Assigns the number of bytes (8) up to the 5th character position in the character string "ABCDEF" to variable K.</p>
<b>Description</b>	<p>The specified <i>character string</i> may contain a mixture of 1-byte and 2-byte characters. The KPOS function returns the number of bytes up to the specified <i>character position</i> in the <i>character string</i>.</p> <p>Specify the position of a character in the character string with the <i>character position</i> parameter.</p> <p>The function returns 0 if the number of characters in the <i>character string</i> is less than the <i>character position</i>.</p>

**KTYPE**

**Kanji TYPE**

(Function)

- Action** Determines the type of character at a specified position in the character string.
- Format** KTYPE (*character string*, *character position*)
- Example** T=KTYPE (K\$, 3)  
Determines the type of the 3rd character in the *character string* and assigns the corresponding number to variable T.
- Description** The KTYPE function determines the type of character at a specified *character position* in the *character-string* as a value between 0 and 2. The meanings of the returned numbers is shown below.

Returned value	Description
0	1-byte alphanumeric standard-sized character
1	2-byte double-sized character
2	2-byte standard-sized character

Specify the character string as a character constant or a character variable containing a mixture of 1-byte and 2-byte characters.

Specify the position of the required character from the start of the *character string* as an integer with the *character position* parameter. Specify the *character position* between 0 and the length of the *character string*. Count both 1-byte and 2-byte characters as one character.

The KLEN function can be used to determine the number of characters in a *character string*.

**LABEL**

**LABELing**

(Command)

- Action** Carries out labelling based on the detailed runlength data measured with the MEASURE command.
- Format** LABEL [*link evaluation constant*]
- Example** LABEL  
Labels using eight-neighbor evaluation of the linked status.
- Description** The LABEL command carries out labelling based on the detailed runlength data measured with the MEASURE command. Labelling data is read with the LDATA function or LPOINT function.  
The RMODE command and MEASURE command must be executed each time before the LABEL command is executed.  
The specified *link evaluation constant* specifies the link evaluation method, as follows:  
0: eight-neighbor evaluation  
1: four-neighbor evaluation  
The default value is 0.  
The following commands and functions are related to the labelling carried out with the LABEL command:

LPUTIMG      LSORT  
LDATA        LPOINT

**LBOUND****Lower BOUNDary**

(Function)

<b>Action</b>	Determines the lower boundary of an array dimension qualifier.
<b>Format</b>	LBOUND ( <i>array name</i> [, <i>number of dimensions</i> ])
<b>Example</b>	I=LBOUND ( A , 1 )  Assigns the lower limit of the qualifier of the 1-dimensional array A to variable I.
<b>Description</b>	The LBOUND function returns the lower boundary of an array dimension qualifier.  Specify the name of the array for which the qualifier is to be determined with the <i>array name</i> parameter.  Specify the number of dimensions of the array with the <i>number of dimensions</i> parameter. The default value is 1.  The returned value is either 0 or 1, as set when the OPTION BASE command was executed.

**LCASE\$****Lower CASE\$**

(Function)

<b>Action</b>	Converts uppercase letters in the character string to-lower case letters.
<b>Format</b>	LCASE\$ ( <i>character string</i> )
<b>Example</b>	INPUT "string:" ,A\$ IF LCASE\$(A\$)="end" THEN END  If the <i>character string</i> input as A\$ is "END", the character string is converted to "end."
<b>Description</b>	The LCASE\$ function converts uppercase letters in the <i>character string</i> to-lower case letters. Existing lowercase letters remain unchanged.

**LDATA****Label DATA**

(Function)

<b>Action</b>	Measures data for the labelled image obtained with the LABEL command.
<b>Format</b>	LDATA ( <i>label#</i> , <i>item</i> )
<b>Example</b>	X=LDATA ( 2 , 1 )  Assigns the value of the center of gravity in the X direction of the image labelled #2 to variable X.
<b>Description</b>	The LDATA function measures data from an image labelled with the LABEL command.  Specify the number of the labelled image as the <i>label#</i> . The label numbers ( <i>label#</i> ) start from 1.

Specify the *item* with one of the following numbers:

- 0: Area
- 1: Center of gravity X
- 2: Center of gravity Y
- 3: Main axis angle
- 4: Peripheral length
- 5: Area after filling
- 6: Number of holes
- 7: X coordinate of top-left corner of external box
- 8: Y coordinate of top-left corner of external box
- 9: X coordinate of bottom-right corner of external box
- 10: Y coordinate of bottom-right corner of external box

The LABEL command must be executed before the LDATA function is used.

## LEFT\$

## LEFT\$

(Function)

<b>Action</b>	Extracts a character string with the specified length from the left end of the specified character string.
<b>Format</b>	LEFT\$ ( <i>character string</i> , <i>character string length</i> )
<b>Example</b>	B\$=LEFT\$ (A\$, 3) Extracts 3 characters from the left end of character string A\$ and assigns them to variable B\$.
<b>Description</b>	Extracts a character string of any length from the start of the specified <i>character string</i> . The <i>character string</i> can be specified as a character constant or character variable. A null string (" ") cannot be specified. Specify the length of the extracted character string in bytes with the <i>character string length</i> parameter as a value between 1 and the length of the <i>character string</i> . A null string is returned if 0 is specified for the <i>character string length</i> . The entire specified character string is returned if the <i>character string length</i> is greater than the length of the specified character string.

## LEN

## LENgth

(Function)

<b>Action</b>	Determines the number of bytes in a character string.
<b>Format</b>	LEN ( <i>character string</i> )
<b>Example</b>	L=LEN("F300 OVL") Assigns the length of the character string "F300 OVL" (8 bytes) to variable L.
<b>Description</b>	The LEN function returns the length of a specified <i>character string</i> in bytes. Use the KLEN function to return the length of a string containing 2-byte characters.

## LET

## LET

(Command)

<b>Action</b>	Assigns the expression at the right to the variable at the left.
<b>Format</b>	[LET] <i>variable</i> = <i>expression</i>

<b>Example</b>	LET A=(B+C)/2 <b>or</b> A=(B+C)/2 Assigns the sum of the numeric variables B and C divided by 2 to variable A.
<b>Description</b>	Assigns the expression at the right to the variable at the left. It is not possible to specify a variable at the right or an expression at the left.  The function name LET can be omitted. LET is normally omitted in a program.  The <i>variable</i> can be a numeric variable or a character variable. Similarly, the <i>expression</i> can be a numeric expression or a character expression. However, the types must match on the left and right. If a numeric variable is specified at the left, a numeric expression must be specified to the right. Similarly, if a character variable is specified at the left, a character expression must be specified to the right.

**LEVEL****LEVEL**

(Command)

<b>Action</b>	Sets the binary level for each binary image plane.
<b>Format</b>	LEVEL <i>binary image plane#</i> , <i>lower limit</i> , <i>upper limit</i> [, <i>mode</i> ]
<b>Example</b>	LEVEL 3,100,200  Set the binary level limits for the binary image plane 3 to 100 and 200, respectively.
<b>Description</b>	The LEVEL command sets the binary level for each binary image plane.  Specify the binary image plane for which the level is set with the <i>binary image plane#</i> parameter. Set the <i>binary image plane#</i> to -1 to set all binary image planes to the same binary level.  The gradations of the raw input image are converted to a binary image with all values between the specified <i>lower limit</i> and <i>upper limit</i> represented as 1.  Set the binary conversion mode with the mode parameter. The new binary level is calculated by a logical operation on the present binary level. The present setting is cancelled if the mode parameter is omitted. Before setting a value with the LEVEL command, cancel the setting that has been made.  OR: set the specified range to 1 NOT: set the specified range to 0 XOR: reverse the specified range

**LINE****LINE**

(Command)

<b>Action</b>	Draws a straight line in VRAM.
<b>Format</b>	LINE X1, Y1, X2, Y2, VRAM [, [page#] [,drawing density <b>or</b> drawing mode]]
<b>Example</b>	LINE 45,35,200,250,2,,255  Draws a straight line with drawing density 255 in the window memory between the start point (45, 35) and end point (200, 250).

**Description** The LINE command draws a straight line between the start and end points. Specify the *VRAM* where the line is drawn with a number, as follows:

- 0: Character memory
- 1: Graphic memory
- 2: Window memory
- 3: Image memory
- 4: Shading memory

Omit the *page#* or set to 0.

The *drawing density* parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The *drawing density* parameter has the following effect when set for the character or graphic memory:

- 0 : 0 written to memory
- 1: 1 written to memory

When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

**LINE INPUT**

**LINE INPUT**

(Command)

**Action** Assigns a line of data input from the keyboard to a character variable.

**Format** LINE INPUT [*prompt character string*{, *or* ;}] *character variable*

**Example** LINE INPUT "DATA" ;A\$

Displays "DATA" on the screen and waits for keyboard input of the line of character data to be assigned to the character variable A\$.

**Description** Operation waits for data input from the keyboard when the LINE INPUT command is executed. Enter the line of data and press the Return Key to assign the data to the variable.

Specify a character string for the *prompt character string*. This character string is displayed to prompt for input of the required data. The *prompt character string* must be separated from the subsequent character variable by a semicolon (;). The specified *prompt character string* only is displayed when the LINE INPUT command is executed.

Input the *character variable* as up to 255 1-byte characters, blanks, commas (,), double quotation marks ("), and numbers. All characters entered from the time the prompt is displayed until the Return Key is pressed are handled as a single character string.

If the Return Key is pressed without entering characters, a null string (") is assigned to the *character variable*.



**LINE INPUT WAIT****LINE INPUT WAIT**

(Command)

<b>Action</b>	Inputs one line of data from the keyboard with a time limitation.
<b>Format</b>	LINE INPUT WAIT <i>wait time</i> , [ <i>prompt character string</i> ]; <i>character variable</i>
<b>Example</b>	LINE INPUT WAIT 70, "ANSWER" ;AN\$ Displays the input prompt message "ANSWER" on the screen and waits 7 seconds for keyboard input of the character data to be assigned to the character variable AN\$.
<b>Description</b>	Operation waits for the specified <i>wait time</i> for data input from the keyboard when the LINE INPUT WAIT command is executed. Enter the line of character data and press the Return Key within the specified <i>wait time</i> to assign the data to the <i>character variable</i> . The LINE INPUT WAIT command is identical to the LINE INPUT command, except for the time limit. If the Return Key is pressed without entering characters, a null string ("") is assigned to <i>character variable</i> . If the time specified by <i>wait time</i> passes, the variables will not change. Specify the <i>wait time</i> in units of 0.1 second.

**LINE INPUT#****LINE INPUT#**

(Command)

<b>Action</b>	Reads a line of data from a sequential access file and assigns it to a character variable.
<b>Format</b>	LINE INPUT # <i>file#</i> , <i>character variable</i>
<b>Example</b>	LINE INPUT #1, L\$ Reads a line of data from file #1 and assigns the data to character variable L\$.
<b>Description</b>	Specify the <i>file#</i> as the number in which the random access file was opened with the OPEN statement. All characters including blanks, commas (,), double quotation marks ("), and numbers between the first character read (including a blank) and the carriage return (CHR\$(13)) are read to the <i>character variable</i> . The LINE INPUT# command can be used in combination with the FOR ... NEXT statements to read consecutive lines of a file up to each carriage return (CHR\$(13)) to successive <i>character variables</i> . The LINE INPUT# command is ideal for reading a line of data up to the carriage return (CHR\$(13)) from a random access file containing both character and numeric variables delimited by blanks or commas (,). Unlike the LINE INPUT command, the LINE INPUT# command displays no prompt message.

**LIST****LIST**

(Command)

<b>Action</b>	Displays all or part of the program contents.
<b>Format</b>	LIST [ <i>line# 1</i> ] [ <i>-line# 2</i> ]
<b>Example</b>	LIST 100-190 Displays the program between line 100 and line 190 on the screen.
<b>Description</b>	Displays the program between <i>line# 1</i> and <i>line# 2</i> on the screen. The entire program is displayed if the <i>line# 1</i> and <i>line# 2</i> parameters are omitted. Only <i>line# 1</i> is printed if <i>line# 2</i> is not specified. Lines from the start of the program to <i>line# 2</i> are printed if <i>line# 1</i> is not specified.

Use a period (.) instead of *line# 1* or *line# 2* to specify the current program line. The current line is indicated by the pointer. The current line is the last line input during program creation or the last line displayed with the LIST command.

**LNUM** **Label NUMBER**

(Function)

- Action** Determines the number of labelled images.
- Format** LNUM
- Example** LN=LNUM  
Assigns the number of labelled images to variable LN.
- Description** The LNUM functions determines the number of images labelled with the LABEL command.  
The LABEL command must be executed before the LNUM function is used.  
-1 is returned if over 255 labels exist.

**LOAD** **LOAD**

(Command)

- Action** Loads a program from disk to memory.
- Format** LOAD *file name* [, R]
- Example** LOAD "C:PRO1"  
Loads the file PRO1 from the memory card.
- Description** The LOAD command loads the file specified with the *file name* parameter. The currently loaded program is deleted from memory.  
If the R option is specified, the program is run with the currently open files immediately it is loaded.  
Use the MERGE command to load a program without deleting the currently loaded program.

**LOC** **LOCation**

(Function)

- Action** Determines the current I/O position within the specified file.
- Format** LOC (*file#*)
- Example** L=LOC ( 1 )  
Assigns the current position of I/O operation in *file#1* to variable L.
- Description** Specify the *file#* as the number in which the file was opened with the OPEN statement.  
The position returned by the LOC function depends on the type of file, as defined in the table below.

File Type	Value returned by LOC
Random access file	Returns the last record# read or written with the GET# or PUT# command as an integer value.
Sequential access file	Returns the number of records read or written since the file was opened as an integer value.
RS-232C	Returns the number of bytes remaining in the communication buffer as an integer value.

**LOCATE****LOCATE**

(Command)

**Action** Sets the cursor position on the text display and whether the cursor is displayed.**Format** LOCATE [*column*] [, *line*] [, *cursor display switch*]**Example** LOCATE 5, 20

Moves the cursor position to the character 5 position (6th character from the left) on line 20 (21st line from the top).

**Description** The LOCATE command moves the cursor on the text display and turns the cursor display on or off.

Specify the horizontal (X) coordinate with the column parameter between 0 and 63. The default value is 0.

Specify the vertical (Y) coordinate with the line parameter between 0 and 24. If this value is omitted, the cursor remains in the line position when the LOCATE command was executed.

Specify the cursor display switch parameter as 0 or 1. The meaning of this setting is shown in the table below. If this value is omitted, the cursor display status when the LOCATE command was executed is maintained.

Cursor display switch parameter	Description
0	No cursor is displayed on the screen.
1	A cursor is displayed on the screen.

**Note** It is not possible to omit the X coordinate, Y coordinate, and cursor display switch with a LOCATE command.**LOF****Length Of File**

(Function)

**Action** Determines the size of a file.**Format** LOF (*file#*)**Example** SIZE=LOF ( 1 )Assigns the size of *file#1* open for I/O operation to variable SIZE.**Description** Specify the *file#* as the number in which the file was opened with the OPEN statement.

The size returned by the LOF function depends on the type of file, as defined in the table below.

File Type	Value returned by LOF
Random access file	Returns the file size as the maximum record number.
Sequential access file	Returns the file size as a number of bytes.
RS-232C	Returns the free space in the communication buffer as a number of bytes.

**LOG****LOGarithm**

(Function)

**Action** Determines the natural logarithm of a number.**Format** LOG (*numeric expression*)**Example** A=LOG ( B )

Assigns the natural logarithm of variable B to variable A.

<b>Description</b>	The LOG function returns the natural logarithm (to base $e = 2.71828$ ) of the <i>numeric expression</i> parameter. The <i>numeric expression</i> must be specified as a positive value.  The <i>numeric expression</i> may be in the form of an integer, long integer, or a single or double-precision real number. The LOG function always returns a double-precision real number.  The EXP function has the opposite action to the LOG function.
--------------------	---

**LPOINT****Label POINT**

(Function)

<b>Action</b>	Determines the label number of a labelled image at a specified position.
<b>Format</b>	LPOINT (X, Y)
<b>Example</b>	N=LPOINT ( 128 , 256 )  Assigns the label number from the labelled image at coordinates (128, 256) to variable N.
<b>Description</b>	The LPOINT function finds the label number at a specified coordinate position from the label number data obtained with the LABEL command  The function returns 0 if no labelled image or a hole in a labelled image exists at the specified coordinates.  The LABEL command must be executed before the LPOINT command is used.

**LPUTIMG****Label PUT IMaGe**

(Command)

<b>Action</b>	Draws a labelled image in VRAM.
<b>Format</b>	LPUTIMG <i>label#</i> , VRAM [, [ <i>page#</i> ] [, <i>plane#</i> ]]
<b>Example</b>	LPUTIMG 2 , 3 , , 2  Draws the image with label #2 to plane 2 of the image memory.
<b>Description</b>	The LPUTIMG command draws labeled image data obtained with the LABEL command to VRAM.  Specify the label number of the image to be drawn with the <i>label#</i> parameter.  Specify the VRAM where the image is drawn with a number, as follows:  0: Character memory 1: Graphic memory 2: Window memory 3: Image memory 4: Shading memory  Omit the <i>page#</i> or set to 0.  Specify the <i>plane#</i> when writing to a frame memory. Specify -1 to write the image to all planes. The image is not written to planes write protected with the MASKBIT command.  The LABEL command must be executed before the LPUTIMG command is used.

**LSET**

**Left SET**

(Command)

<b>Action</b>	Writes left-justified character data to a variable area defined with the FIELD command.
<b>Format</b>	LSET <i>character variable</i> = <i>character string</i>
<b>Example</b>	LSET A\$="BASIC" Left justifies the character string "BASIC" and writes it to the variable area defined with the variable name A\$.
<b>Description</b>	Specify a character variable name defined with the FIELD command with the <i>character variable</i> parameter. Specify a character constant or character variable as the character string. Excess characters are lost from the right of the character string if the length of the specified <i>character string</i> exceeds the length of the <i>character variable</i> defined with the FIELD command. Conversely, if the length of the specified <i>character string</i> is less than the length of the <i>character variable</i> , the remaining positions are filled with blanks.

**LSORT**

**Label SORT**

(Command)

<b>Action</b>	Renumbers labels in order of area.
<b>Format</b>	LSORT <i>mode</i>
<b>Example</b>	LSORT 0 Renumbers the label numbers in descending order of area.
<b>Description</b>	The LSORT command sorts the labeled image data obtained with the LABEL command into order of area. Specify the <i>mode</i> parameter to 0 to sort the label numbers in descending order or to 1 to sort the label numbers in ascending order. The LABEL command must be executed before the LSORT command is used.

**LTRIM\$**

**Left TRIM\$**

(Function)

<b>Action</b>	Deletes spaces to the left of a character string.
<b>Format</b>	LTRIM\$ ( <i>character string</i> )
<b>Example</b>	A\$=LTRIM\$ (" JAPAN" ) Assigns "JAPAN" to variable A\$.
<b>Description</b>	Returns the <i>character string</i> (1- or 2-byte characters) with the spaces removed from the left.

**MASKBIT**

**MASKBIT**

(Command)

<b>Action</b>	Disables writing to specific planes in the frame memory.
<b>Format</b>	MASKBIT VRAM, [ <i>page#</i> ], <i>bit data</i>
<b>Example</b>	MASKBIT 2 , , &HFF Disables writing to the window memory.

<b>Description</b>	<p>The MASKBIT command enables or disables writing to each plane of the frame memory.</p> <p>Specify the <i>VRAM</i> with the <i>VRAM</i> parameter, as follows:</p> <ul style="list-style-type: none"> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the plane to be write disabled with the <i>bit data</i> parameter. Set the bit corresponding to a plane to 1 to disable writing or to 0 to enable writing.</p>
--------------------	---

**MDATA**

**Measure DATA**

(Function)

<b>Action</b>	Reads the measured results, such as area, center of gravity, and axis angle.
<b>Format</b>	MDATA ( <i>binary image plane#</i> , <i>measured data</i> )
<b>Example</b>	<p>A=MDATA ( 0 , 2 )</p> <p>Assigns the center of gravity X coordinate of the binary image in plane 0 in pixels to variable A.</p>
<b>Description</b>	<p>The MDATA function reads the area, center of gravity, and the axis angle results measured with the MEASURE command.</p> <p>Specify the <i>binary image plane#</i> as the plane number from which the data is read.</p> <p>Specify the <i>measured data</i> with one of the values listed below to determine the the type of measured data to read. The calibration is based on the current scene data, which is specified with the CHANGE command.</p> <ul style="list-style-type: none"> <li>0: Area (pixel units)</li> <li>1: Area after calibration</li> <li>2: Center of gravity X (pixel units)</li> <li>3: Center of gravity X after calibration</li> <li>4: Center of gravity Y (pixel units)</li> <li>5: Center of gravity Y after calibration</li> <li>6: Axis angle (pixel units)</li> <li>7: Axis angle after calibration</li> </ul> <p>The MMODE and MEASURE commands must be executed before the MDATA function is used.</p>

**MDATA2**

**Measure DATA2**

(Function)

<b>Action</b>	Reads the edge angle measurement results.
<b>Format</b>	MDATA2 ( <i>binary image plane#</i> , <i>X array</i> , <i>Y array</i> , <i>function</i> )
<b>Example</b>	<p>A=MDATA2 ( 0 , X , Y , 0 )</p> <p>Assigns the edge angle in the rectangular region defined by X array and Y array on binary image plane 0 to variable A.</p>
<b>Description</b>	<p>The MDATA2 function reads the edge angle results measured with the MEASURE command.</p> <p>Specify the top-left coordinates of the rectangle where the edge angle is to be determined with <i>X array</i> and the bottom-right coordinates with <i>Y array</i>. These coordinates are matched to the rectangular window.</p>

Set the function parameter to 0 to determine uncalibrated (pixel) data or to 1 to determine calibrated data.

The calibration is based on the current scene data, which is specified with the CHANGE command.

The MMODE command must be used to turn on the run function and the MEASURE command executed before the MDATA2 function is used.

## **MEASURE**

## **MEASURE**

(Command)

<b>Action</b>	Measures the area, center of gravity, axis angle, simple run length, and detailed run length.
<b>Format</b>	MEASURE
<b>Example</b>	MEASURE Measures under the conditions set using MMODE or RMODE.
<b>Description</b>	The MEASURE command measures the area, center of gravity, axis angle, simple run length, and detailed run length under the conditions set using MMODE and RMODE command.  Set the measurement conditions with the MMODE command before measuring the area, center of gravity, or axis angle. Turn on the run function and set the measurement conditions with the MMODE command before measuring the simple run length.  Set the measurement conditions with the RMODE command before measuring the detailed run length.

## **MENU**

## **MENU**

(Command)

<b>Action</b>	Returns to the Menu mode.
<b>Format</b>	MENU
<b>Example</b>	MENU Returns to the Menu mode.
<b>Description</b>	Quits the OVL mode and returns to the Menu mode.

## **MERGE**

## **MERGE**

(Command)

<b>Action</b>	Merges a program from the memory card with the program in memory.
<b>Format</b>	MERGE <i>filename</i>
<b>Example</b>	MERGE "C:PRO1" Merges the program PRO1 from the memory card.
<b>Description</b>	Merges the program currently loaded in memory with the program specified by the filename parameter from the memory card. The program is reorganized into line number order after merging.  Existing lines in memory are deleted if line numbers are duplicated in the programs in memory and in the memory card.  The program cannot be executed automatically after merging. Use the CHAIN command to automatically execute programs.

**MID\$****MIDdle\$**

(Function, Command)

<b>Action</b>	Extracts part of a character string. Changes part of a character string, if required.
<b>Format</b>	Format 1 (function): MID\$( <i>character variable</i> , <i>start position</i> [, <i>number of characters</i> ]) Format 2 (command): MID\$( <i>character variable</i> , <i>start position</i> [, <i>number of characters</i> ]) = <i>character string</i>
<b>Example</b>	MID\$(A\$, 6, 8) = "F300 OVL" Replaces the 8 characters starting from the 6th character with the character string "F300 OVL" (8 characters).
<b>Description</b>	<u>Format 1</u> Extracts the specified <i>number of characters</i> from the <i>start position</i> of the <i>character variable</i> . Specify the position of the first character in the <i>character variable</i> to be extracted as a value between 1 and the number of characters in the <i>character variable</i> . Specify the number of characters to be extracted with the <i>number of characters</i> parameter. All characters to the right of the <i>start position</i> are extracted if the <i>number of characters</i> is omitted or if the <i>number of characters</i> exceeds the actual number of characters between the <i>start position</i> and the right end of the <i>character variable</i> . <u>Format 2</u> Replaces the specified <i>number of characters</i> from the start position of the <i>character variable</i> with the <i>character string</i> . Specify the position of the first character in the <i>character variable</i> to be replaced as a value between 1 and the number of characters in the <i>character variable</i> . Specify the number of characters to be replaced with the <i>number of characters</i> parameter. All characters to the right of the <i>start position</i> are replaced if the <i>number of characters</i> is omitted or if the number of characters exceeds the actual number of characters between the <i>start position</i> and the right end of the <i>character variable</i> . Specify the <i>character string</i> as a character constant or a character variable.

**MKD\$****MaKe Double\$**

(Function)

<b>Action</b>	Converts a double-precision value to internal character-type representation.
<b>Format</b>	MKD\$ ( <i>double-precision value</i> )
<b>Example</b>	C\$=MKD\$ (C#) Converts the specified double-precision real number variable C# to a character string and assigns it to variable C\$.
<b>Description</b>	Because numeric data cannot be handled in a random access file, the double-precision numeric data must be converted to character-type numeric data with the MKD\$ function before it can be written to a random access file. This character-type numeric data is then converted to a character string corresponding to the internal representation (binary representation) of the number type, which is used in the random access file. The MKD\$ function converts a <i>double-precision real number</i> to an 8-byte character string. The CVD function reverts a character-type <i>double-precision number</i> converted with the MKD\$ function to a numeric value.



**MKDIR****MaKe DIRectory**

(Command)

<b>Action</b>	Creates a new directory for the memory card.
<b>Format</b>	MKDIR <i>directory name</i>
<b>Example</b>	MKDIR "BASIC" Creates a new directory named "BASIC" under the current directory.
<b>Description</b>	Specify the name of the new directory name with the <i>directory name</i> parameter. An error occurs if a directory with the specified name already exists in the same path. The new directory is created under the current directory if no new path is specified with the <i>directory name</i> . If a new path is specified with the <i>directory name</i> , the new directory is created outside the current directory. Use the ¥ symbol to delimit directories.

**MKI\$****MaKe Integer\$**

(Function)

<b>Action</b>	Converts an integer value to internal character-type representation.
<b>Format</b>	MKI\$ ( <i>integer</i> )
<b>Example</b>	A\$=MKI\$ (A%) Converts the specified integer variable A% to a character string and assigns it to variable A\$.
<b>Description</b>	Because numeric data cannot be handled in a random access file, the integer data must be converted to character-type numeric data with the MKI\$ function before it can be written to a random access file. This character-type numeric data is then converted to a character string corresponding to the internal representation (binary representation) of the number type, which is used in the random access file. The MKI\$ function converts an integer to a 2-byte character string. The CVI function reverts a character-type integer number converted with the MKI\$ function to a numeric value.

**MKL\$****MaKe Long \$**

(Function)

<b>Action</b>	Converts a long integer value to internal character-type representation.
<b>Format</b>	MKL\$ ( <i>long integer</i> )
<b>Example</b>	A\$=MKL\$ (A&) Converts the specified long integer variable A& to a character string and assigns it to variable A\$.
<b>Description</b>	Because numeric data cannot be handled in a random access file, the <i>long integer</i> data must be converted to character-type numeric data with the MKL\$ function before it can be written to a random access file. This character-type numeric data is then converted to a character string corresponding to the internal representation (binary representation) of the number type, which is used in the random access file. The MKL\$ function converts a <i>long integer</i> to a 4-byte character string. The CVL function reverts a character-type long integer number converted with the MKL\$ function to a numeric value.

**MKS\$****MaKe Single\$**

(Function)

<b>Action</b>	Converts a single-precision value to internal character-type representation.
<b>Format</b>	MKS\$ ( <i>single-precision value</i> )
<b>Example</b>	B\$=MKS\$ ( B! ) Converts the specified single-precision real number variable B! to a character string and assigns it to variable B\$.
<b>Description</b>	Because numeric data cannot be handled in a random access file, the single-precision numeric data must be converted to character-type numeric data with the MKS\$ function before it can be written to a random access file. This character-type numeric data is then converted to a character string corresponding to the internal representation (binary representation) of the number type, which is used in the random access file. The MKS\$ function converts a <i>single-precision real number</i> to a 4-byte character string. The CVS function reverts a character-type <i>single-precision number</i> converted with the MKS\$ function to a numeric value.

**MMODE****Measure MODE**

(Command)

<b>Action</b>	Sets the basic measurement mode for each binary image plane.
<b>Format</b>	MMODE <i>binary image plane#</i> , <i>measured pixels</i> [, [ <i>window function</i> ] [, [ <i>paint function</i> ] [, [ <i>fill function</i> ] [, [ <i>run function</i> ]]]]]
<b>Example</b>	MMODE 0,0,1 Sets measurement of black pixels for binary image plane 0, with the window function on.
<b>Description</b>	The MMODE command sets the conditions for measurement with the MEASURE command for each of the 8 binary image planes. Specify the binary image plane for which the measurement conditions are set with the <i>binary image plane#</i> parameter. Set the <i>binary image plane#</i> to -1 to set the conditions for all binary image planes simultaneously. Set the <i>measured pixels</i> parameter to set which pixels to measure, as follows: 0: Black pixels 1: White pixels Normally set the <i>window function</i> parameter to 1. Set the <i>paint function</i> parameter to set the paint and pattern matching functions, as follows: 0: No painting or pattern matching 1: Carry out painting 2: Carry out pattern matching The default value is 0. Set the <i>fill function</i> parameter to set the contour measurement, as follows: 0: Normal All of the white pixels in the window will be taken as the object for measurement. 1: Fill All of the white pixels between the starting and ending points along the horizontal axis will be measured. The default value is 0.

Set the *run function* parameter to set the simple run length data measurement, as follows:

- 0: Do not measure the simple run length data
  - 1: Measure the simple run length data
- The default value is 0.

**NAME****NAME**

(Command)

<b>Action</b>	Renames files stored in the memory card.
<b>Format</b>	<code>NAME old filename AS new filename</code>
<b>Example</b>	<code>NAME "ABC" AS "DEF"</code> Renames the file name ABC as the file name DEF.
<b>Description</b>	Specify the old filename as the name of an existing file in the memory card as a character string. An error occurs if the specified filename does not exist.  Specify the new name of the file with the new filename parameter. Use only 1-byte characters. An error occurs if a file with the same name as the specified new filename already exists.  A filename cannot be renamed with the NAME command if the file attribute is set to write-protected with the SET command. Use the SET command to remove the write protection before renaming the file with the NAME command.

**NEW****NEW**

(Command)

<b>Action</b>	Deletes the program from memory.
<b>Format</b>	<code>NEW</code>
<b>Example</b>	<code>NEW</code> Deletes the program from memory.
<b>Description</b>	The NEW command deletes the program from memory and resets numeric constants to 0 and character constants to a null string (""). Any open I/O files are closed. The Direct mode is selected after the NEW command is executed.

**NPIECE****Number of PIECE**

(Function)

<b>Action</b>	Determines the number of smaller character strings the specified character string is divided into by the delimiters.
<b>Format</b>	<code>NPIECE (character string, delimiter)</code>
<b>Example</b>	<code>A=NPIECE("OMRON F300", " ")</code> Because the space character is set as the delimiter, OMRON and F300 are considered as separate character strings, so 2 is assigned to variable A.
<b>Description</b>	The NPIECE function determines the number of small character strings separated by the <i>delimiters</i> in the specified <i>character string</i> . Returns the number of character strings divided by the delimiters. The function returns 1 if the <i>delimiter</i> character is not contained in the <i>character string</i> .

## OCT\$

## OCTal\$

(Function)

- Action** Converts a decimal number to an octal character string.
- Format** OCT\$ (*numeric expression*)
- Example** A\$="&O"+OCT\$(10)  
 Converts the decimal value 10 to the octal character string 12, concatenates this character string with the character string "&O", and assigns the result (&O12) to the character variable A\$.
- Description** The OCT\$ function converts a decimal value to an octal character string. It does not add the "&O" prefix to indicate an octal character string.  
 Specify the *numeric expression* as a decimal numeric constant or numeric variable between  $-2^{31}$  and  $2^{31}-1$ . Any decimal places in the specified *numeric expression* are rounded off to create an integer which is converted to the octal character string.  
 The relationship between the decimal *numeric expression* and octal character string is as follows:

Numeric expression (decimal)	Octal character string
-2147483648 to -1	"20000000000" to "37777777777"
0 to 2147483647	"0" to "17777777777"

To convert an octal character string back to numeric data, append the "&O" prefix to indicate an octal character string, then convert the character string with the VAL function.

## ON COM GOSUB SUBroutine

## ON COMMunication GO to

(Command)

- Action** Defines the jump destination of the interrupt subroutine when data is received at the communication port.
- Format** ON COM (*RS-232C port#*) GOSUB *line# or label*
- Example** ON COM(1) GOSUB \*PROCESS  
 Defines the jump destination of the interrupt subroutine when data is received by the RS-232C channel 0 as the label name \*PROCESS.
- Description** Specify the *RS-232C port#* as 1 or 2. The default value is 1. The ON COM GOSUB commands define the jump destination of the interrupt subroutine executed when data is received at the specified communication port. Interrupt operation starts when it is enabled by the COM ON command. Operation returns when the RETURN command is executed.

```

10 OPEN "COM:" AS #1
20 ON COM GOSUB *LABEL1
30 COM ON
   |
   |
50 *LABEL1
   |
90 RETURN
    
```

} Interrupt subroutine

**ON ERROR GOTO****ON ERROR GOTO**

(Command)

<b>Action</b>	Defines the first line of the error processing routine executed when an error occurs.
<b>Format</b>	ON ERROR GOTO <i>line# or label or 0</i>
<b>Example</b>	ON ERROR GOTO 1000 Defines the jump destination when an error occurs as line# 1000.
<b>Description</b>	<p>The ON ERROR ... GOTO command defines the jump destination of the error processing routine executed when an error occurs. When an error occurs, operation jumps to the specified jump destination and the error routine is executed to reset the error.</p> <p>Specify the first line of the error processing routine with the <i>line# or label</i> parameter.</p> <p>Specify <i>0</i> instead of the <i>line# or label</i> to cancel the error processing routine jump destination defined previously with the ON ERROR GOTO command. If an error subsequently occurs, an error message is displayed and program operation stops. When an error occurs, the error code and line number where the error occurred are assigned to the system variables ERR and ERL.</p> <p>Operation returns to the original program when the RESUME command at the end of the error processing routine is executed.</p>

**ON HELP GOSUB****ON HELP key GO to SUBroutine**

(Command)

<b>Action</b>	Defines the first line of the interrupt subroutine executed when the HELP Key is pressed.
<b>Format</b>	ON HELP GOSUB <i>line# or label</i>
<b>Example</b>	ON HELP GOSUB *SHORI Defines the jump destination when the HELP Key is pressed as the label name *SHORI.
<b>Description</b>	<p>The <i>line# or label</i> parameter defines the jump destination of the interrupt subroutine executed when the HELP Key is pressed.</p> <p>The interrupt subroutine defined by the <i>line# or label</i> parameter is executed when the HELP Key is pressed during program execution if the interrupt operation is enabled with the HELP ON command.</p> <p>The ON HELP GOSUB command is one type of key input interrupt routine. It is used as an aid to program operation by explaining how to use the program and input data.</p> <p>Operation returns to the original program when the RETURN command at the end of the interrupt subroutine is executed. Operation can be transferred to a different position from the position where the interrupt occurred by specifying a <i>line# or label</i> parameter with the RETURN command.</p>

**ON INTR GOSUB****ON INTeRrupt GO to SUBroutine**

(Command)

<b>Action</b>	Defines the first line of the interrupt subroutine executed when a STEP interrupt occurs.
<b>Format</b>	ON INTR GOSUB <i>line# or label</i>
<b>Example</b>	ON INTR GOSUB 2000 Defines the first line of the interrupt subroutine executed when a STEP interrupt occurs as line# 2000.

**Description** The ON INTR GOSUB command defines the line# or label of the first line of the interrupt subroutine executed when a STEP interrupt occurs. Any further STEP interrupt occurring during execution of the interrupt subroutine is ignored.

## ON KEY GOSUB ON function KEY GO to SUBroutine

(Command)

**Action** Defines the first line of the interrupt subroutine executed when a function key is pressed.

**Format** ON KEY GOSUB *line# or label* [, *line# or label...*]

**Example** ON KEY GOSUB , , \*PROCESS

Defines the jump destination when a function key (F3) is pressed as the label name \*PROCESS.

**Description** The *line#* or *label* parameter defines the jump destination of the interrupt subroutine executed when a function key is pressed.

Up to ten *line#* or *label* parameters can be specified delimited by commas (.). The order of the *line#* or *label* parameters corresponds to the function key numbers.

The interrupt subroutine defined by the *line#* or *label* parameter is executed only when the corresponding function key is pressed during program execution.

The ON KEY GOSUB command is one type of key input interrupt routine. Defining frequently executed routines or frequently input data in the interrupt subroutines makes the function keys a convenient aid to programming.

Operation returns to the original program when the RETURN command at the end of the interrupt subroutine is executed. Operation can be transferred to a different position from the position where the interrupt occurred by specifying a *line#* or *label* parameter with the RETURN command.

## ON STOP GOSUB ON STOP key GO to SUBroutine

(Command)

**Action** Defines the first line of the interrupt subroutine executed when the STOP Key is pressed.

**Format** ON STOP GOSUB *line# or label*

**Example** ON STOP GOSUB \*PROCESS

Defines the jump destination when the STOP Key is pressed as the label name \*PROCESS.

**Description** The *line#* or *label* parameter defines the jump destination of the interrupt subroutine executed when the STOP Key is pressed.

The interrupt subroutine defined by the *line#* or *label* parameter is executed when the STOP Key is pressed during program execution if the interrupt operation is enabled with the STOP ON command.

The ON STOP GOSUB command is one type of key input interrupt routine. Defining how to stop a specified operation in the interrupt subroutine at the specified jump destination allows the operation to be stopped by pressing the STOP Key.

Operation returns to the original program when the RETURN command at the end of the interrupt subroutine is executed. Operation can be transferred to a different position from the position where the interrupt occurred by specifying a *line#* or *label* parameter with the RETURN command.

**ON TIME\$ GOSUB                      ON TIME\$ GO to SUBroutine**

(Command)

<b>Action</b>	Defines the time when the timer interrupt is generated and the first line of the interrupt subroutine.
<b>Format</b>	ON TIME\$ = "HH:MM:SS" GOSUB <i>line# or label</i>
<b>Example</b>	ON TIME\$="02:00:00" GOSUB 1000 Sets the interrupt to be generated at 02:00:00 and defines the first line of the interrupt subroutine executed at this time as line# 1000.
<b>Description</b>	<p>The ON TIME\$ GOSUB command sets the time when the timer interrupt is generated and defines the <i>line# or label</i> of the first line of the interrupt subroutine executed at the set time. If branching is enabled with the TIME\$ ON command, the specified interrupt routine is executed when the set time is reached.</p> <p>Set the time in the format: <i>HH:MM:SS</i>. Set the hour between 00 and 23 and set the minutes and seconds between 00 and 59.</p> <p>Example 1: ON TIME\$ = "01:00:00" GOSUB 1000 ... interrupt generated at 1 am.</p> <p>Example 2: ON TIME\$ = "10:30:00" GOSUB 1000 ... interrupt generated at 10.30 am.</p> <p>The <i>HH:MM:SS</i> set with this command does not affect the setting of the internal clock.</p> <p>The <i>line# or label</i> parameter defines the first line of the interrupt subroutine.</p> <p>Operation returns to the original program when the RETURN command at the end of the interrupt subroutine is executed. Operation can be transferred to a different position from the position where the interrupt occurred by specifying a <i>line# or label</i> parameter with the RETURN command.</p> <p>The time setting is 00:00:00 after the F300 power is turned on.</p>

**ON GOSUB****ON GO to SUBroutine**

(Command)

<b>Action</b>	Branches program operation according to specified conditions.
<b>Format</b>	ON <i>expression</i> GOSUB [ <i>line# or label</i> ] [, <i>line# or label...</i> ]
<b>Example</b>	ON S GOSUB *STARTRTN, *ENDRTN, *ERRRTN Branches to subroutine *STARTRTN when the variable S equals 1, to *ENDRTN when S equals 2, and to *ERRRTN when S equals 3.
<b>Description</b>	<p>Program operation branches to subroutines specified by a line# or label according to the value of the expression.</p> <p>The subroutine defined with the first <i>line# or label</i> parameter is executed when the expression equals 1, the subroutine defined with the second <i>line# or label</i> parameter is executed when the expression equals 2, and so on.</p> <p>Program operation flows to the next line if the expression equals 0 or if the line# and label parameters are omitted.</p> <p>When the RETURN command at the end of the subroutine is executed, program operation jumps back to the line after the ON GOSUB command. If a <i>line# or label</i> parameter is specified with the RETURN command, operation jumps to the specified line.</p>

**ON GOTO****ON GO TO**

(Command)

<b>Action</b>	Branches program operation according to specified conditions.
<b>Format</b>	ON <i>expression</i> GOTO [ <i>line# or label</i> ] [, <i>line# or label...</i> ]
<b>Example</b>	ON T GOTO 200,500,900  Branches to line 200 when the variable T equals 1, to line 500 when T equals 2, and to 900 when T equals 3.
<b>Description</b>	Program operation branches to the line specified by a <i>line#</i> or <i>label</i> according to the value of the expression. Operation jumps to the line defined with the first <i>line#</i> or <i>label</i> parameter when the expression equals 1, the second <i>line#</i> or <i>label</i> parameter when the expression equals 2, and so on.  Program operation flows to the next line if the expression equals 0 or if the <i>line#</i> and <i>label</i> parameters are omitted.

**OPEN ( 1 )****OPEN**Command

<b>Action</b>	Opens a file.
<b>Format</b>	OPEN <i>filename</i> [FOR { <i>OUTPUT or INPUT or APPEND</i> }] AS <i>file#</i>
<b>Example</b>	OPEN "DATA01" FOR OUTPUT AS #1  Opens the sequential access file DATA01 in the memory card as file#1 to output data to the file.
<b>Description</b>	A sequential access file or random access file in the memory must be opened using the OPEN command with a specified <i>file#</i> before file I/O operations are possible. Use the CLOSE command to close the file after I/O operations are complete.  When opening a sequential access file, the I/O mode must be specified, as one of the following:  INPUT ... read data from the file OUTPUT ... write data to the file APPEND ... append data to an existing file  Do not specify the I/O mode when opening a random access file. If the I/O mode is specified, the random access file is treated as a sequential access file with the same name, so that an error occurs when I/O operations are attempted with the GET# and PUT# commands.  Specify the <i>file#</i> as a positive integer between 1 and 13. The same <i>file#</i> cannot be applied to more than one open file simultaneously. The <i>file#</i> assigned to the file can be used instead of the <i>filename</i> for all I/O operations until the file is closed with the CLOSE command.  Use the INPUT# and LINE INPUT# commands and the INPUT\$ function to read data from a sequential access file and the PRINT#, PRINT# USING, and WRITE# command to write data to the sequential access file.  Define the file buffer with the FIELD command before I/O operations with a random access file. Use the LSET and RSET commands to set data in the file buffer and the PUT# and GET# commands to read and write data to and from the file.



**OPEN ( 2 )**

**OPEN**

(Command)

- Action** Opens the RS-232C port.
- Format** OPEN "COM[RS-232C port#]: [baud rate [parity [data length [stop bits [XON switch]]]]]" [FOR OUTPUT or INPUT] AS [#] file#
- Example** OPEN "COM:" FOR OUTPUT AS #1  
Opens the RS-232C channel 0 for data output.
- Description** Specify the RS-232C port# as 1 or 2, as follows:  
1: channel 0  
2: channel 1

The default value is 1.

The OPEN command opens the RS-232C port for data I/O. When data I/O is complete, close the RS-232C port with the CLOSE command.

Set the *baud rate*, *parity*, *data length*, *stop bits*, and *XON switch* parameters to match the specifications of the device communicated with. Refer to the following table for details of these settings. The default value shown in the table is automatically set if the parameter is omitted.

Parameter	Setting	Description	Default value
Baud rate	1200	1,200 bps	9600
	2400	2,400 bps	
	4800	4,800 bps	
	9600	9,600 bps	
	19200	19,200 bps	
Parity	E	Even parity check	N
	O	Odd parity check	
	N	No parity check	
Data length	7	Each character 7 bits	8
	8	Each character 8 bits	
Stop bits	1	One stop bit	1
	2	Two stop bits	
XON switch	X	XON/XOFF control enabled	---
	---	XON/XOFF control disabled	
S parameter	S	ON	N
	N	OFF	

**OPTION BASE**

**OPTION BASE**

(Command)

- Action** Declares minimum value of the array qualifier.
- Format** OPTION BASE 0 or 1
- Example** OPTION BASE 0  
Declares the minimum value of the array qualifier as 0.
- Description** The OPTION BASE command declares the minimum value of the array qualifier as 0 or 1.  
The qualifier minimum value is automatically set to 0 if an array variable is declared with the DIM command in a program containing no OPTION BASE

command. The OPTION BASE command is used to set the qualifier minimum value to 1. After setting the qualifier minimum value with the OPTION BASE command, do not execute the OPTION BASE command again in the same program to change the minimum value.

**PIECE\$****PIECE\$**

(Function)

<b>Action</b>	Extracts partial character strings divided by the delimiter characters from the specified character string.
<b>Format</b>	PIECE\$ ( <i>character string, delimiter character string</i> [, <i>start#</i> [, <i>end##</i> ])
<b>Example</b>	A\$=PIECE\$("F300 OVL" , " " , 2) Assigns the second partial character string "OVL" from the character string "F300 OVL" to variable A\$.
<b>Description</b>	The PIECE\$ function returns the partial character strings between the specified <i>start#</i> and <i>end #</i> divided by the <i>delimiter characters</i> from the specified character string.  If the <i>end#</i> is omitted, the single partial string corresponding to the <i>start#</i> is returned. The first partial character string is returned if both the <i>start#</i> and <i>end#</i> are omitted. The entire character string is returned if it is not divided by the specified delimiter character.

**PIN****Port IN**

(Function)

<b>Action</b>	Reads the bit status of a specified bit of the Terminal Block Unit or Parallel I/O Unit input port.
<b>Format</b>	PIN ( <i>bit address</i> )
<b>Example</b>	A=PIN(12) Assigns the bit status of bit 12 of the Terminal Block Unit or Parallel I/O Unit input port to variable A.
<b>Description</b>	The PIN function reads the bit status of the Terminal Block Unit or Parallel I/O Unit input port bit specified with the <i>bit address</i> parameter. The function returns a value, as follows: <ul style="list-style-type: none"> <li>1: bit status ON</li> <li>2: bit status OFF</li> </ul> If both Terminal Block Units and Parallel I/O Units are connected, the PIN function does not differentiate between the two. The bit status can be specified up to the total number of input bits for all units.  <i>Bit address</i> 0 is the lowest terminal number of the lowest numbered slot.

**POINT****POINT**

(Function)

<b>Action</b>	Determines the density of the specified coordinates in VRAM.
<b>Format</b>	POINT ( <i>X, Y, VRAM</i> [, <i>page#</i> ])
<b>Example</b>	G=POINT(128, 255, 3, 0) Assigns the density at image memory coordinates (128, 255) to variable G.

<b>Description</b>	<p>The POINT function reads the density at the specified coordinates (X, Y) in the specified VRAM.</p> <p>Specify the VRAM with the VRAM parameter, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>The function returns either 0 or 1 if a plane VRAM is specified or a value between 0 and 255 if a frame VRAM is specified.</p> <p>Omit the <i>page#</i> or set to 0.</p>
--------------------	--

**POLYGON****POLYGON**

(Command)

<b>Action</b>	Draws a polygon in VRAM.
<b>Format</b>	POLYGON <i>number of data</i> , <i>X array</i> , <i>Y array</i> , <i>VRAM</i> [, [ <i>page#</i> ] [, [ <i>drawing density or drawing mode</i> ] [, <i>linear</i> ]]]
<b>Example</b>	<p>POLYGON 28, XD, YD, 3, ,128,0</p> <p>Draws a polygon with drawing density 128 in image memory with the points at the coordinates defined in the arrays XD and YD with 28 data items.</p>
<b>Description</b>	<p>The POLYGON command draws a polygon with points at the coordinates in the X array and Y array from the start of the array to the number of points specified with the number of data parameter.</p> <p>Specify the number of data as a value up to 64.</p> <p>Specify the VRAM where the polygon is drawn with a number, as follows:</p> <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> <p>Omit the <i>page#</i> or set to 0.</p> <p>Specify the drawing method with the <i>drawing density</i> or the <i>drawing mode</i>. The default value is <i>drawing mode</i>, OR.</p> <p>The <i>drawing density</i> parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The <i>drawing density</i> parameter has the following effect when set for the character or graphic memory:</p> <ul style="list-style-type: none"> <li>0 : 0 written to memory</li> <li>1: 1 written to memory</li> </ul>

**POLYLINE****POLYLINE**

(Command)

<b>Action</b>	Draws a polyline in VRAM.
<b>Format</b>	POLYLINE <i>number of data</i> , <i>X array</i> , <i>Y array</i> , <i>VRAM</i> [, [ <i>page#</i> ] [, [ <i>drawing density or drawing mode</i> ]]]
<b>Example</b>	<p>POLYLINE 28, XD, YD, 2, ,128</p> <p>Draws a polyline with drawing density 128 in window memory plane 7 with the points at the coordinates defined in the arrays XD and YD with 28 data items.</p>

**Description** The POLYLINE command draws a polyline with points at the coordinates in the *X array* and *Y array* from the start of the array to the number of points specified with the *number of data* parameter.

Specify the *number of data* as a value up to 64.

Specify the *VRAM* where the polyline is drawn with a number, as follows:

- 0: Character memory
- 1: Graphic memory
- 2: Window memory
- 3: Image memory
- 4: Shading memory

Omit the *page#* or set to 0.

Specify the drawing method with the *drawing density* or the *drawing mode*. The default value is *drawing mode*, OR.

When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

**POS**

**POSiTion**

(Function)

**Action** Determines the current cursor column position.

**Format** POS (*numeric expression*)

**Example** HP%=POS(0)

Assigns the current cursor column position (0 to 63) to the variable HP%.

**Description** The POS function determines the position of the character cursor in the X direction (along the line).

The parameter of the POS function is a dummy. It has no special significance, but cannot be omitted. Normally, set the dummy parameter to 0.

Use the CSRLIN function to determine the line where the cursor is positioned.

**POUT**

**Port OUT**

(Command)

**Action** Controls the bit status of a specified bit of the Terminal Block Unit or Parallel I/O Unit output port.

**Format** POUT *bit address*, 0 or 1

**Example** POUT 2,1

Turns ON bit 2 of the Terminal Block Unit or Parallel I/O Unit output port.

**Description** The POUT function controls the bit status of the Terminal Block Unit or Parallel I/O Unit output port bit specified with the *bit address* parameter.

If both Terminal Block Units and Parallel I/O Units are connected, the POUT function does not differentiate between the two. The bit status can be specified up to the total number of input bits for all units. Specify the bit address of the bit to control as a value between 0 and (maximum bit address – 1).

Bit address 0 is the lowest terminal number of the lowest numbered slot.

**PRINT**

**PRINT**

(Command)

**Action** Displays data on the text display.

**Format** PRINT [*expression*] [; or , [*expression*].] [; or ,]

**Example** PRINT A\$,B\$

Displays the character data defined by the character expressions A\$ and B\$ sequentially on the text display.

**Description** The *expression* parameter can be specified as a character expression or a numeric expression.

Delimit multiple *expressions* by commas (,) or semicolons (;). The display depends on the delimiter character used, as follows:

Comma (,)  
 14 characters are automatically allocated for each *expression*. An *expression* less than 14 characters in length is separated from the next expression by blanks.

Semicolon (;)  
 The *expressions* are displayed closed together. No fixed data length is allocated to the *expressions*. Numeric data is displayed preceded by a space for the sign and followed by a blank delimiter character.

Character returns are added automatically under the conditions described below.

Only a character return is displayed if all *expression* parameters are omitted. If no semicolon (;) is included after the final *expression* parameter, the character return is added after all the *expressions* are displayed. If a semicolon (;) is included after the final *expression* parameter, no character return is added after all the *expressions* are displayed and the first *expression* specified with a subsequent PRINT command is displayed consecutively.

A character return is added if the number of spaces remaining in the displayed line is less than the number of characters in the specified *expression*. Character data is displayed in the spaces remaining in the displayed line.

**PRINT USING**

**PRINT USING**

(Command)

**Action** Displays formatted data on the text display.

**Format** PRINT USING *format control character string* ; *expression* [{; or ,} *expression* ...] [; or ,]

**Example** PRINT USING "& &###" ; A\$, B

Displays the character data and numeric data defined by the character expression A\$ and numeric data B sequentially on the text display, with 4 characters for A\$ and 3 characters for B.

**Description** Specify the *format control character string* in double quotations ("). The specified characters themselves are not displayed on the text display. The specified *format control character string* determines the number of characters and format in the display of the *expressions*.

Specify the character string defining character data or numeric string defining numeric data with the *expression* parameters.

Delimit multiple *expressions* by commas (,) or semicolons (;). The delimiter character used does not affect the display.

The format control character string may contain two types of characters: one type to control the display of character data and the other type to control the display of numeric data. The specified format control character string characters must match the type of data displayed. The control characters are described in the following tables.

**Format Control Character Strings for Displaying Character Data**

Character	Description
!	Display only the first character of the character string.
&(n blanks)&	The number of characters displayed is defined by the number of blanks (n) contained between the ampersands (&). A total of (n+2) characters is displayed from the start of the character string. The remaining characters are ignored if the character string contains more than (n+2) characters. The remaining character spaces are filled with blanks if the character string contains less than (n+2) characters.
@	Display the character string unchanged.
_	The character following the underscore character is not only a format control character and is printed.

**Format Control Character Strings for Displaying Numeric Data**

Character	Description
#	The number of # characters specifies the number of numeric data digits, including the +/- sign. If the specified number of characters is less than the number of numeric characters, the displayed data is right justified with blanks to the left.
.	Insert the decimal point (.) in combination with the # characters to specify the decimal position. Redundant decimal places are filled with zeros.
+	Specify the + sign as a prefix or suffix of the format control character string to print the + sign before or after the numeric data. If two or more + signs are specified consecutively, the second (and subsequent) + sign is considered to be outside the format control character string.
-	Specify the - sign as a suffix of the format control character string, the minus sign for negative numeric data is output to the right of the number (i.e., after the number) If two or more - signs are specified consecutively, the second (and subsequent) - sign is considered to be outside the format control character string.
**	Specify two or more asterisks (*) at the start of the format control character string to output asterisks instead of blanks to the left of the numeric data. The asterisks (*) fill all blanks in the specified number of characters.
¥¥	Specify two or more yen signs (¥) at the start of the format control character string to output a yen sign in the blank space to the left of the numeric data. The two yen signs (¥) occupy two character spaces and one of these is used to print the yen sign.
**¥	Specify two or more asterisks (*) plus a single yen sign (¥) at the start of the format control character string to output asterisks instead of blanks to the left of the numeric data and a yen sign in the blank space immediately to the left of the numeric data. The two asterisks (*) and the yen sign (¥) occupy three character spaces and one of these is used to print the yen sign.
,	Use a comma (,) in combination with the # number characters to delimit each 3 digits of the integer portion of the number. If the comma (,) is specified to the right of the decimal point (.), the comma (,) is output after the numeric data.
^ ^ ^ ^	Specify four carets (^) after the # number characters to output the number in exponential format.
_	The character following the underscore character is not only a format control character and is printed.

A non-control character in the format control character string is displayed before or after the character or numeric data.

A percent sign (%) is displayed in front of the numeric data if the digits of the numeric data format specified with the expression parameters exceed the size of the region specified by the format control characters.

**PRINT #**

**PRINT #**

(Command)

- Action**                      Writes data to a sequential access file.
- Format**                        PRINT# file# [, expression[ { ; or , } expression ... ] ] ; or ,
- Example**                        PRINT #1 , A\$ , B\$

Organizes the character data defined by the character expressions A\$ and B\$ into sequential data in the order A\$, B\$, and writes the data to file#1.

**Description** Specify the *file#* as the number in which the sequential access file was opened for output with the OPEN statement. After writing to the sequential access file is complete, the file must be closed with the CLOSE statement.

Except for the fact that the PRINT# command writes data to a file instead of displaying data on the screen, the command is identical to the PRINT command. Refer to the PRINT command for information on specifying the expressions, on the meaning of the delimiter characters, and on the rules covering the carriage return characters.

## PRINT# USING

## PRINT# USING

(Command)

**Action** Writes formatted data to a sequential access file.

**Format** PRINT *file#*, USING *format control character string* ; *expression* [{; **or** } *expression ...*] [**or** ,]

**Example** PRINT #1, USING "& &###" ; A\$, B

Organizes the data defined by the character expression A\$, formatted as 4 characters, and the numeric data B, formatted as 3 characters, into sequential data in the order A\$, B, and writes the data to file#1.

**Description** Specify the *file#* as the number in which the sequential access file was opened for output with the OPEN statement. After writing to the sequential access file is complete, the file must be closed with the CLOSE statement.

Except for the fact that the PRINT# USING command writes data to a file instead of displaying data on the screen, the command is identical to the PRINT USING command. Refer to the PRINT USING command for information on specifying the *format control character string* and expressions, on the meaning of the delimiter characters, and on the rules covering the carriage return characters.

## PSET

## Point SET

(Command)

**Action** Draws a point in VRAM.

**Format** PSET X, Y, VRAM [, [*page#*] [, *drawing density or drawing mode*]]

**Example** PSET 100, 300, 2, , 128

Draws a point with drawing density 128 in the window memory at coordinates (100, 300).

**Description** The PSET command draws a point at the specified coordinates.

Specify the *VRAM* where the point is drawn with a number, as follows:

- 0: Character memory
- 1: Graphic memory
- 2: Window memory
- 3: Image memory
- 4: Shading memory

Omit the *page#* or set to 0.

Specify the drawing method with the *drawing density* or the drawing mode. The default value is *drawing mode*, OR.

The *drawing density* parameter specifies the density between 0 and 255 when drawing to the window, image, or shading memory. The default value is 255. The *drawing density* parameter has the following effect when set for the character or graphic memory:

- 0 : 0 written to memory
- 1 : 1 written to memory

The *drawing mode* settings operate as follows:

OR: The current contents of the image memory ORed with 255 are written to memory.

NOT: 0 is written to memory

XOR: The current contents of the image memory are inverted.

The default value for the *drawing mode* is OR.

When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

## PUT #

## PUT #

(Command)

<b>Action</b>	Writes data from the file buffer to a random access file.
<b>Format</b>	PUT # file# [, <i>numeric expression</i> ]
<b>Example</b>	PUT #1 , 8  Writes data from the file buffer to the #8 record in the random access file opened as file #1.
<b>Description</b>	The PUT# command writes the data stored in the file buffer to the random access file specified with the file#. Specify the file# as the number in which the random access file was opened with the OPEN statement. Specify the number of the record to be written to the file with the <i>numeric expression</i> . If omitted, the record after the record# used with the previous GET# or PUT# command is read.

## PUT@

## PUT@

(Command)

<b>Action</b>	Draws the array variable data stored in the image memory to VRAM.
<b>Format</b>	PUT@ X, Y, <i>array name</i> , VRAM [, [ <i>page#</i> ] [, <i>plane#</i> ]]
<b>Example</b>	PUT@ 100 , 300 , A , 2  Draws the contents of the array A to a rectangular region of the window memory with top-left coordinates (100, 300).
<b>Description</b>	The PUT@ command draws (overwrites) data from the specified array to a rectangular area with the top-left coordinates (X, Y). Specify the VRAM where the data is drawn with a number, as follows: <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Window memory</li> <li>3: Image memory</li> <li>4: Shading memory</li> </ul> Omit the <i>page#</i> or set to 0. Specify the <i>plane number</i> if a frame VRAM is specified with the VRAM parameter (2 to 4). If a frame VRAM is specified, the contents of planes write protected with the MASKBIT command remain unchanged.



## RANDOMIZE

## RANDOMIZE

(Command)

<b>Action</b>	Initializes random number generation.
<b>Format</b>	RANDOMIZE [ <i>numeric expression</i> ]
<b>Example</b>	RANDOMIZE 10 Sets the seed value to initialize random number generation to 10.
<b>Description</b>	The RANDOMIZE command sets a new seed value to initialize random number generation. The RANDOMIZE command only initializes random number generation; use the RND function to generate random numbers.  Specify the seed value with the <i>numeric expression</i> between -32768 to 32767. If the <i>numeric expression</i> is omitted, a message is displayed and the system waits for a seed value between -32768 to 32767 to be input from the keyboard.

## RDATA

## Runlength DATA

(Function)

<b>Action</b>	Reads the detailed run length data.
<b>Format</b>	RDATA ( <i>XY coordinate, run#, measured data</i> )
<b>Example</b>	N=RDATA ( 0 , 2 , 2 ) Assigns the edge start coordinate of the #2 run at XY coordinate position 0.
<b>Description</b>	The RDATA command reads measurement data based on the detailed run length data obtained with the MEASURE command.  Specify the coordinates where the run data (line length) is to be read. Set the Y coordinate if the RMODE measuring direction was set to the X direction, or set the X coordinate if the measuring direction was set to the Y direction.  Specify the number of the run data to be read at the specified XY coordinate position with the <i>run#</i> parameter. The function returns 0 if the specified <i>run#</i> exceeds the number of existing runs. The <i>run#</i> starts from 0.  Specify the type of data to be read from the specified run with the <i>measured data</i> parameter, as follows:  <ul style="list-style-type: none"> <li>0: Number of runs</li> <li>1: Run length</li> <li>2: Edge coordinate (start point)</li> <li>3: Edge coordinate (end point)</li> </ul> The <i>run#</i> is ignored if the <i>measure data</i> is set to 0.

## READ

## READ

(Command)

<b>Action</b>	Reads data defined with the DATA statement to a variable.
<b>Format</b>	READ <i>variable</i> [, <i>variable</i> ...]
<b>Example</b>	READ A , B Reads data to the numeric variables A and B.
<b>Description</b>	The READ command reads the data defined with the DATA statement to sequentially specified variables. Consequently, the READ command must be used with the DATA statement.  Specify the variables separated by commas (,). The format of the variables must match the format specified with the DATA statement.

If the number of variables specified for the READ command is less than the number specified for the DATA statement, the remaining data is read by the next READ command. The remaining data is ignored if no further READ command is specified. The number of variables specified to be read must not exceed the number specified with the DATA statement. Take care when programming successive READ commands.

Execute the RESTORE command before the READ command to specify the line containing the DATA statement to read. If no RESTORE command is used before the READ command, the READ command reads the next data after the previously read data (if any), otherwise it reads the data specified with the first DATA statement in the program. Therefore, it is possible for READ commands on different lines of the program to read the same data several times.

**REM****REMark**

(Command)

<b>Action</b>	Inserts remarks into the program.
<b>Format</b>	{REM <i>or</i> , } <i>remark</i>
<b>Example</b>	REM PROGRAM NAME PROGRAM01 <i>or</i> 'PROGRAM NAME PROGRAM01
<b>Description</b>	The REM statement is non-executable and has not effect on the operation of the program. It is used to insert remarks and explanations into the program text. A single-quotation mark (') can be used instead of the REM statement. All characters and symbols specified as the parameter are handled as a remark. Commands and functions included in the remark are not executed. A colon (:) is also considered as part of the remark and cannot be used to continue the line onto the next line.

**RENUM****RENUMber**

(Command)

<b>Action</b>	Renumbers the lines in all or part of the program.
<b>Format</b>	RENUM [ <i>new line#</i> ] [, [ <i>old line#</i> ] [, <i>increment</i> ]]
<b>Example</b>	RENUM 1000,100 Renumbers the current line 100 as line 1000 and renumbers all subsequent lines to the end of the program in increments of 10.
<b>Description</b>	The RENUM command renumbers the <i>old line#</i> as the <i>new line#</i> and renumbers all subsequent lines to the end of the program by the specified increment. The default value for the <i>increment</i> is 10. The program is renumbered from the first line if the <i>old line#</i> is omitted. The default value of the <i>new line#</i> is 10.

**REPLACE****REPLACE**

(Command)

<b>Action</b>	Searches for the specified old character string between line# 1 and line# 2 and replaces the it with the new character string.
<b>Format</b>	REPLACE <i>old character string</i> , <i>new character string</i> [, [ <i>line# 1 or label 1</i> ] [–[ <i>line# 2 or label 2</i> ]]]
<b>Example</b>	REPLACE "FALSE" , "TRUE" , 50–100 Replaces the word "FALSE" with "TRUE" between lines 50 and 100.

<b>Description</b>	The REPLACE command replaces the <i>old character string</i> with the <i>new character string</i> between <i>line# 1 (or label 1)</i> and <i>line# 2 (or label 2)</i> and replaces the it with the new character string.  The search and replace operation starts from the first line of the program if the <i>line# 1 (or label 1)</i> parameter is omitted. The search and replace operation continues to the last line of the program if the <i>line# 2 (or label 2)</i> parameter is omitted.
--------------------	---

**RESTORE****RESTORE**

(Command)

<b>Action</b>	Specifies the line with the DATA statement to be read by the READ command.
<b>Format</b>	RESTORE [ <i>line# or label</i> ]
<b>Example</b>	RESTORE 150  Specify the DATA command in line 150 to be read by the READ command.
<b>Description</b>	The RESTORE command specifies the DATA statement to be read by a subsequent READ command with a <i>line# or label</i> . If a <i>line#</i> is specified, the DATA statement in the specified line is read by the READ command. If a <i>label</i> is specified, the DATA statement declared after the label is read.  If the <i>line#</i> and <i>label</i> are omitted, subsequent READ commands read the DATA statement on the line with the smallest line number.

**RESUME****RESUME**

(Command)

<b>Action</b>	Ends an error processing routine and restarts the operation before the error occurred.
<b>Format</b>	RESUME [ <i>line# or label or 0 or NEXT</i> ]
<b>Example</b>	RESUME 1500  Return operation to line 1500.
<b>Description</b>	The RESUME command ends an error processing routine and returns operation to the position where the error occurred. The RESUME command must be executed within an error processing routine.  The <i>line# or label</i> specifies the jump destination after the error processing routine execution is complete. Normally, a <i>line# or label</i> would be specified to repeat the series of processes which led to the error. If the <i>line#</i> and <i>label</i> are omitted or if <i>0</i> is specified, operation returns to the statement where the error occurred.  If <i>NEXT</i> is specified, operation returns to the statement after the statement where the error occurred.

**RETURN****RETURN**

(Command)

<b>Action</b>	Ends a subroutine and returns operation to the position where the subroutine was called or to a specified line.
<b>Format</b>	RETURN [ <i>line# or label</i> ]
<b>Example</b>	RETURN  Returns operation to the line after the line where the subroutine was called.

<b>Description</b>	<p>The RETURN command ends a subroutine and returns operation to the position where the subroutine was called (the line after the GOSUB command) or to a specified line.</p> <p>More than one RETURN command can be contained in a single subroutine.</p> <p>The <i>line#</i> or <i>label</i> can be specified to force operation to jump to the specified position. The specified <i>line#</i> or <i>label</i> must be identical to the label called with the original GOSUB command. Correct operation cannot be guaranteed if the specified <i>line#</i> or <i>label</i> differs from the label specified with the GOSUB command, due to the loss of the correlation between the GOSUB and RETURN commands. Take care when executing a GOSUB command inside a FOR-NEXT loop if GOSUB commands are nested.</p>
--------------------	--

**RIGHT\$****RIGHT\$**

(Function)

<b>Action</b>	Extracts a character string with the specified length from the right end of the specified character string.
<b>Format</b>	RIGHT\$ ( <i>character string</i> , <i>character string length</i> )
<b>Example</b>	<p>B\$=RIGHT\$ ("F300-OVL" , 3)</p> <p>Extracts 3 characters from the right end of character string "F300-OVL" and assigns them to variable B\$.</p>
<b>Description</b>	<p>Extracts a character string of any length from the end of the specified <i>character string</i>.</p> <p>The <i>character string</i> can be specified as a character constant or character variable. A null string (" ") cannot be specified.</p> <p>Specify the length of the extracted character string in bytes with the <i>character string length</i> parameter as a value between 1 and the length of the <i>character string</i>. A null string is returned if 0 is specified for the <i>character string length</i>. The entire specified character string is returned if the <i>character string length</i> is greater than the length of the specified character string.</p>

**RMDIR****ReMove DIRectory**

(Command)

<b>Action</b>	Deletes a directory from the memory card.
<b>Format</b>	RMDIR <i>directory name</i>
<b>Example</b>	<p>RMDIR "BASIC"</p> <p>Deletes the directory named BASIC from the current directory.</p>
<b>Description</b>	<p>Specify the <i>directory name</i> as the name of a directory in the memory card with a character string. An error occurs if the specified directory does not exist.</p> <p>The directory specified with the <i>directory name</i> cannot be deleted if it contains files. The files in the directory must be deleted with the KILL command before using the RMDIR command.</p> <p>Do not specify the current directory as the <i>directory name</i> parameter for the RMDIR command. To delete the current directory, first use the CHDIR command to change the current directory to another directory (normally one directory up the hierarchy) before using the RMDIR command.</p>

**RMODE****Runlength MODE**

(Command)

<b>Action</b>	Sets the measurement mode for detailed run length data.
<b>Format</b>	RMODE <i>binary image plane#</i> , <i>measured pixels</i> [, [ <i>window function</i> ] [, [ <i>direction</i> ] [, [ <i>noise</i> ] [, <i>image cutout</i> ]]]]
<b>Example</b>	RMODE 3,0,1,0  Sets measurement of black pixels for binary image plane 3, with the window function on, X-direction measurement, no noise filtering, and image cutout turned off.
<b>Description</b>	<p>The RMODE command sets the conditions for measurement of the detailed runlength with the MEASURE command. Specify the binary image plane for which the measurement conditions are set with the <i>binary image plane#</i> parameter.</p> <p>Set the <i>measured pixels</i> parameter to set which pixels to measure, as follows:</p> <ul style="list-style-type: none"> <li>0: Black pixels</li> <li>1: White pixels</li> </ul> <p>Set the <i>window function</i> to specify measurement in the window, as follows:</p> <ul style="list-style-type: none"> <li>0: Measure the whole screen.</li> <li>1: Measure in window only</li> </ul> <p>Set the <i>direction</i> parameter to specify the measurement direction of the detailed run length data, as follows:</p> <ul style="list-style-type: none"> <li>0: X direction</li> <li>1: Y direction</li> </ul> <p>The default value is 0.</p> <p>Measurement must be made in the X direction before measuring in the Y direction.</p> <p>Set the <i>noise</i> parameter to specify a number of pixels between 0 and 3 to ignore as noise. This number or less of consecutive pixels is not considered as run length data.</p> <p>Set the <i>image cutout</i> to 1 to recognize all parts outside the window as part of the run.</p> <p>The RMODE command must be executed before the MEASURE command.</p>

**RND****RaNDom**

(Function)

<b>Action</b>	Generates a random number between 0 and 1.
<b>Format</b>	RND [( <i>numeric expression</i> )]
<b>Example</b>	A=RND(1)  Assigns the next random number in the current random number sequence to variable A%.

**Description** Returns a random number between 0 and 1. The random number is generated with 1 as the random number seed until the seed is changed with the RANDOMIZE command.

The returned random number depends on the specified *numeric expression*. The meaning of the specified *numeric expression* is described in the table below.

Value	Description
Negative value	Returns the first random number from the current random number sequence.
0	Returns the previous random number in the current random number sequence.
Positive value or omitted	Returns the next random number in the current random number sequence.

## RSET

## Right SET

(Command)

**Action** Writes right-justified character data to a variable area defined with the FIELD command.

**Format** RSET *character variable* = *character string*

**Example** RSET A\$="BASIC"  
Right justifies the character string "BASIC" and writes it to the variable area defined with the variable name A\$.

**Description** Specify a character variable name defined with the FIELD command with the *character variable* parameter.

Specify a character constant or character variable as the *character string*.

Excess characters are lost from the left of the character string if the length of the specified *character string* exceeds the length of the *character variable* defined with the FIELD command. Conversely, if the length of the specified *character string* is less than the length of the *character variable*, the remaining positions are filled with blanks.

## RTRIM\$

## Right TRIM\$

(Function)

**Action** Deletes spaces to the right of a character string.

**Format** RTRIM\$ (*character string*)

**Example** A\$=RTRIM\$ ("JAPAN ")  
Assigns "JAPAN" to variable A\$.

**Description** Returns the character string (1- or 2-byte characters) with the spaces removed from the right.

## RUN

## RUN

(Command)

**Action** Runs a program.

**Format** Format 1: RUN [*line#*]  
Format 2: *filename* [, R]

**Example** RUN "PRO1"  
Load and execute the program PRO1 on the memory card.

**Description** The command in Format 1 runs the program from the specified *line#*. The program runs from the first line if the *line#* is omitted.

The command in Format 1 loads the program with the specified *filename* from the memory card and runs if from the first line. If the R option is specified, files previously opened for I/O are kept open.

The RUN signal turns ON when the RUN command is executed.

**RUNL****RUN Length**

(Function)

<b>Action</b>	Reads the simple run length data.
<b>Format</b>	RUNL ( <i>binary plane#</i> , <i>Y coordinate</i> , <i>measured data</i> )
<b>Example</b>	L=RUNL( 2 , 128 , 0 ) Assigns the simple runlength data from Y coordinate 128 of the binary image plane 2 to variable L.
<b>Description</b>	The RUNL command reads the simple run length data obtained for each binary image plane with the MMODE command run function parameter set to 1. Set the number of the binary image plane for which the data is required with the <i>binary plane#</i> parameter. Specify the Y coordinate of the run data to be read with the <i>Y coordinate</i> parameter. Specify the type of data to be read with the <i>measure data</i> parameter, as follows: <ul style="list-style-type: none"> <li>0: Run length</li> <li>1: Start X coordinate</li> <li>2: End X coordinate</li> <li>3: Error data (The function returns -1 if more than one run exists.)</li> </ul> Set the MMODE command run function parameter set to 1 before using the RUNL function.

**SAVE****SAVE**

(Command)

<b>Action</b>	Saves an OVL program from memory to the memory card.
<b>Format</b>	SAVE <i>filename</i> [, <i>P</i> ]
<b>Example</b>	SAVE "MYPROG" Save the OVL program in memory with the filename MYPROG.BAS.
<b>Description</b>	The SAVE command saves a program in memory in ASCII format to the file specified with the filename in the memory card or RS-232C port. If the name of an existing file is specified as the <i>filename</i> , the old contents of the file are deleted. The file is coded if the P option is specified. An error (Illegal function call) occurs if the LOAD, LIST, or EDIT command is executed on a file saved using the P option. An error (Access denied) also occurs if the OPEN command is executed on a file saved using the P option.

**SBANK****Shading memory BANK**

(Command)

<b>Action</b>	Selects the shading memory bank number.
<b>Format</b>	SBANK ( <i>bank#</i> )
<b>Example</b>	SBANK 0 Switches to shading memory bank 0.

**Description** The SBANK command switches to the shading memory bank 0 or 1 specified with the *bank#*.

**SCAN**

**SCAN**

(Command)

**Action** Conducts scan measurement of a binary image.

**Format** `SCAN VRAM, [page#], [plane#], measured pixels [, [window function] [, [NOISE], [, [X-coordinate compensation] [, [Y-coordinate compensation]]]]`

**Example** `SCAN 3, , 5, 0, 0, , 20, -15`  
 Sets measurement of black pixels for binary image plane 5, with the window function on, no noise filtering, +20 X coordinate compensation, and -15 Y coordinate compensation.

**Description** The SCAN command conducts scan measurement of a binary image under the conditions specified by the SCANSET command. Read the measured results using the SDATA1 and SDATA2 functions.  
 Specify the *VRAM* to be scanned with a number, as follows:  
 0: Character memory  
 1: Graphic memory  
 2: Window memory  
 3: Image memory  
 4: Shading memory  
 Omit the *page#* or set to 0.  
 The *plane#* is ignored if a plane VRAM is specified, but the *plane#* must be set if a frame VRAM is specified.  
 Set the *measured pixels* parameter to 0 to measure black pixels or 1 to measure white pixels.  
 Set the *window function* to 1 scan the specified region of the image or to 0 to scan the entire screen. The default value is 1.  
 Set the noise parameter to specify a number of pixels between 0 and 3 to ignore as noise. This number or less of consecutive pixels is not considered as scan data. The default value is 0.  
 Specify the X-coordinate compensation or Y-coordinate compensation parameter to a displacement offset value between -256 and 255. The default value is 0.

**SCANSET**

**SCANSET**

(Command)

**Action** Sets the conditions for scan measurement.

**Format** `SCANSET shape, number of array data elements, X array, Y array`

**Example** `SCANSET 0, 2, X, Y`  
 Sets scan measurement around an ellipse.

**Description** The SCANSET commands sets the conditions for scan measurement.  
 Specify the *shape* of the series of pixels to be scanned with the *shape* parameter, as follows:  
 0: Ellipse (including circle)  
 1: Polygonal line  
 2: Polygon



Define the scanned *shape* with the parameter arrays *X array* and *Y array*.

For an ellipse, specify the center coordinates and the X and Y half axes.

X array (0): X coordinate of center Y array (0): Y coordinate of center  
 X array (1): X axis half axis Y array (1): Y axis half axis

For non-ellipse shapes, specify X and Y coordinate series.

X array (0): X coordinate of point 0 Y array (0): Y coordinate of point 0  
 X array (1): X coordinate of point 1 Y array (1): Y coordinate of point 1  
 X array (2): X coordinate of point 2 Y array (2): Y coordinate of point 2  
 to  
 X array (n): X coordinate of point n Y array (n): Y coordinate of point n

Specify the qualifier (i.e., the *number of data elements*) in *X array* and *Y array*.

**SCNCALIB**

**SCeNe CALIBrAtion**

(Command)

<b>Action</b>	Specifies the scene and camera# used for the measured value calibration data.
<b>Format</b>	SCNCALIB <i>scene#</i> , <i>camera#</i>
<b>Example</b>	SCNCALIB 0, 3  Sets the scene 0, camera 3 data as the calibration data used to calculate the calibrated data read with the MDATA function. Specifies the camera# to be calibrated.
<b>Description</b>	The SCNCALIB command sets the scene specified by the <i>scene#</i> and the camera specified by the <i>camera#</i> as the data used for measurement calibration.  The SCNCALIB command is used to specify the scene calibration data set from the menu mode, when calibrated data is read with the MDATA function.  Both the <i>scene#</i> and <i>camera#</i> are set to 0 when OVL is booted up.

**SCNCAM**

**SCeNe CAMeRa**

(Function)

<b>Action</b>	Reads the camera setting data for the specified scene and binary image plane#.
<b>Format</b>	SCNCAM ( <i>scene#</i> , <i>binary image plane#</i> , <i>data type</i> )
<b>Example</b>	L=SCNCAM( 3, 4, 0 )  Assigns the camera number of scene 3, binary plane 4, to variable L.
<b>Description</b>	The SCNCAM function reads the camera setting data for the binary image plane specified by the <i>binary image plane#</i> in the scene specified by the <i>scene#</i> .  Specify the <i>type of data</i> to read with the data type parameter using a number, as follows:  0: Camera number 1: Camera magnification 2: Camera angle in degrees

**SCNLEVEL**

**SCeNe LEVEL**

(Function)

<b>Action</b>	Determines the binary level data for a binary image plane of a specified scene.
<b>Format</b>	SCNLEVEL ( <i>scene#</i> , <i>binary image plane#</i> , <i>data type</i> )
<b>Example</b>	L=SCNLEVEL( 3, 4, 0 )  Assigns the binary level (lower limit) of scene 3, binary plane 4, to variable L.

<b>Description</b>	The SCNLEVEL function reads the binary level data for the binary image plane specified by the <i>binary image plane#</i> in the scene specified by the <i>scene#</i> . Specify the <i>type of binary level data</i> to read with the data type parameter using a number, as follows: 0: Binary level (lower limit) 1: Binary level (upper limit)
--------------------	---

**SCNLOAD****SCeNe LOAD**

(Command)

<b>Action</b>	Loads the scene measuring condition from a file.
<b>Format</b>	SCNLOAD <i>file name</i> , <i>scene#</i>
<b>Example</b>	SCNLOAD "file"  Load the data from the file named "file" in the memory card as the scene data for scene 0.
<b>Description</b>	The SCNLOAD command loads the data from the file specified with the <i>file name</i> parameter as scene data and stores it as the measuring conditions for the scene specified by the <i>scene#</i> .  Scene data can be loaded with the SCNLOAD function only from a file saved with the SCNSAVE command or saved in the menu mode.

**SCNLUT****SCeNe LUT**

(Command)

<b>Action</b>	Sets the binary level of all binary planes in the specified scene to the binary LUT.
<b>Format</b>	SCNLUT <i>scene#</i>
<b>Example</b>	SCNLUT 6  Sets the binary level of all binary planes in the scene #6 to the binary LUT.
<b>Description</b>	The SCNLUT command batch sets the binary level of all binary planes in the scene specified with the <i>scene#</i> to the binary LUT.

**SCNSAVE****SCeNe SAVE**

(Command)

<b>Action</b>	Saves the scene measuring condition data to a file.
<b>Format</b>	SCNSAVE <i>file name</i> , <i>scene#</i>
<b>Example</b>	SCNSAVE "file" , 0  Saves the scene data for scene 0 to the file named "file" in the memory card.
<b>Description</b>	The SCNSAVE command saves the scene data specified with the <i>scene#</i> parameter to the file specified by the <i>file name</i> .

**SDATA1****Scan DATA1**

(Function)

<b>Action</b>	Reads the scan measurement data statistics.
<b>Format</b>	SDATA1 ( <i>data type</i> )
<b>Example</b>	ED=SDATA1 ( 2 )  Assigns the maximum scan length measured by the SCAN command to variable ED.

<b>Description</b>	<p>The SDATA1 function reads the scan measurement data statistics measured with the SCAN command. All values are returned in units of pixels.</p> <p>Specify the statistic required with the data type parameter using one of the following numbers:</p> <ul style="list-style-type: none"> <li>0: Number of scans</li> <li>1: Average scan length</li> <li>2: Maximum scan length</li> <li>3: Minimum scan length</li> <li>4: Median scan length</li> <li>5: Scan number with the maximum length</li> <li>6: Scan number with the minimum length</li> <li>7: Scan number with the median length</li> </ul>
--------------------	---

**SDATA2****Scan DATA2**

(Function)

<b>Action</b>	Reads the individual scan data measurement items.
<b>Format</b>	SDATA2 ( <i>scan#</i> , <i>data type</i> )
<b>Example</b>	<pre>ED=SDATA2 ( 2 , 2 )</pre> <p>Assigns the start point Y coordinate to variable ED.</p>
<b>Description</b>	<p>The SDATA2 function reads the individual scan data measurement items of the specified scan# measured with the SCAN command. All values are returned in units of pixels.</p> <p>Specify the data item.</p> <ul style="list-style-type: none"> <li>0: Scan length</li> <li>1: Start point X coordinate</li> <li>2: Start point Y coordinate</li> <li>3: End point X coordinate</li> <li>4: End point Y coordinate</li> <li>5: Length from scan origin to start point</li> <li>6: Length from scan origin to end point</li> </ul>

**SEARCH****SEARCH**

(Function)

<b>Action</b>	Searches for an integer in an integer array and determines the number of the array element where the integer is found.
<b>Format</b>	SEARCH ( <i>integer array name</i> , <i>value to find</i> [, [ <i>start element#</i> ] [, <i>increment#</i> ]])
<b>Example</b>	<pre>S%=SEARCH ( TE% , 60 )</pre> <p>Searches for the integer 60 in the integer array (100) and assigns the number of the first element where the integer 60 is stored to variable S%.</p>
<b>Description</b>	<p>The SEARCH function finds the specified value in the integer array and returns the the number of the array element where the value is found as an integer. -1 is returned if the specified value is not found in the array.</p> <p>Specify the <i>integer array name</i> as the name of an array variable defined as a one-dimensional array with the DIM command. Only one-dimensional array variables may be specified.</p> <p>Specify the integer to be found with the <i>value to find</i> parameter. To find a single- or double-precision integer value, first convert the value to an integer.</p> <p>Specify the number of array element to start the search with the <i>start element#</i> parameter. Specify any value between the minimum and maximum qualifier val-</p>

ue. The search starts from the start of the array variable if this parameter is omitted. The minimum qualifier value is declared with the OPTION BASE command.

Specify the *increment* as a positive integer. The *increment* sets the counter between the searched array elements. The *increment* is added to the number of a searched element to determine the number of the next element to search. Intermediate elements are not searched. The default value is 1 if the parameter is omitted and all elements after the *start element#* are searched.

## **SELECT...CASE-CASE ELSE-END SELECT**

## **SELECT...CASE-CASE ELSE-END SELECT**

(Command)

**Action** Provides multiple branching depending on the result of a conditional expression.

**Format** SELECT *expression*  
 [CASE *item* [, *item*...]  
           *Statement in CASE block*  
 CASE ELSE  
           *Statement in CASE ELSE block*  
 END SELECT

**Example**

```
SELECT A
CASE 0
    PRINT "ZERO"
CASE 1, 3, 5, 9
    PRINT "ODD"
CASE 2, 4, 6, 8,
    PRINT "EVEN"
END SELECT
```

Branches program execution depending on the value of variable A.

**Description** The SELECT command branches program execution depending on the value of the specified expression.

The *expression* can be defined as a numeric or character expression. The program branches as defined when the value of the expression matches the specified CASE value.

Multiple CASE statements may be defined. The CASE and CASE ELSE statement may be omitted.

The END SELECT statement is required; it must not be omitted.

If more than one CASE statement matches the result of the expression, the first of the CASE statements is executed.

Do not use the GOTO command to jump into or out of the SELECT block.

## **SET**

## **SET**

(Command)

**Action** Sets the write-protect attribute for a file.

**Format** SET *filename* **or** *file#*, *attribute character*

**Example**

```
SET "FILE", "P"
Write protect the file named FILE.

SET #1, " "
Write enable file #1.
```

**Description** Sets the write-protect attribute (write protect or write enable) with the *attribute character* for the file specified with the *filename* or *file#*. The attribute is applied to the specified file only. Other files in the memory card remain unchanged.

Specify the *filename* parameter as the name of an existing file with a character string. After the write-protect attribute is set for a file specified with a filename it remains unchanged until cancelled with the SET command.

Specify the *file#* as the number in which the file was opened with the OPEN command. This attribute is maintained only while the file is open.

Specify the *attribute character* as either P or a null string or single space. Other characters cause an error. The key to the attribute characters is shown in the table below.

Attribute character	Attribute setting
"P"	Writing disabled to the file specified with the filename of file#. No data output to the file with file output commands (PRINT#/PUT#/WRITE#).
Null string or single space	Cancels write protection for the file specified with the filename of file# to enable data to be written to the file.

**SETBLUT**

**SET Binary LUT**

(Command)

**Action** Sets array data as binary LUT data.

**Format** SETBLUT *binary image plane#*, *array name* [, [*qualifier*] [, *size*]]

**Example** SETBLUT 2 , A  
Set the 256 array elements from the start of array A as binary LUT.

**Description** The SETBLUT command sets the array specified by the *array name* as the binary LUT data for the specified *binary image plane#*.

The *qualifier* specifies the first array element to be set as the binary LUT data. The default value is 0.

The *size* parameter specifies the number of array elements, as follows:  
0: 256  
1: 512

The default value is 0.

The array data corresponds to the binary LUT data, as follows:  
0: 0 set to the LUT  
non-0: 1 set to the LUT

**SETDLUT**

**SET Display LUT**

(Command)

**Action** Sets an array variable in the display LUT.

**Format** SETDLUT *region*, *array name*, [, *qualifier*]

**Example** SETDLUT 1 , A  
Sets 256 elements of array A as the display LUT for inside the window.

**Description** The SETDLUT command sets the values in the array variable specified by the *array name* as the display LUT data.

Separate display LUTs are provided for inside and outside the window. Specify the display LUT for inside or outside the window with the *region*, parameter. Enter 0 to set the display LUT for outside the window or 1 for inside the window.

The *qualifier* specifies the first array element to be set as the display LUT. The default value is 0.

**SETDLVL****SET Display LeVeL**

(Command)

<b>Action</b>	Sets the display level for each display image.
<b>Format</b>	SETDLVL <i>image type</i> [, <i>gradation</i> ]
<b>Example</b>	SETDLVL 1,20 Set the graphic memory display level to 20.
<b>Description</b>	The SETDLVL command sets the display brightness for the image specified with the <i>image type</i> parameter. <ul style="list-style-type: none"> <li>0: Character memory</li> <li>1: Graphic memory</li> <li>2: Mask image</li> <li>3: Binary image, white</li> <li>4: Binary image, black</li> <li>5: Window memory increment</li> <li>6: Paint/pattern matching window memory increment</li> </ul> Set the required display level with the <i>gradation</i> parameter. If omitted, the setting reverts to the setting at OVL boot-up.

**SETLUT****SET LUT**

(Command)

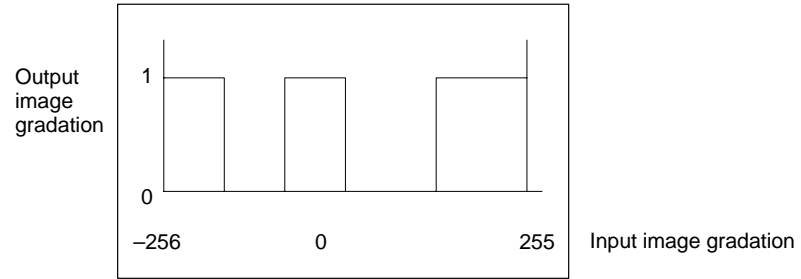
<b>Action</b>	Sets array data as the filter LUT data.
<b>Format</b>	SETLUT <i>array name</i> , [, [ <i>qualifier</i> ] [, <i>size</i> ]]
<b>Example</b>	SETLUT A Sets 256 elements of array A as the filter LUT.
<b>Description</b>	The SETLUT command sets the array data variable specified by the <i>array name</i> as the filter LUT data. <p>The <i>qualifier</i> specifies the first array element to be set as the filter LUT data. The default value is 0.</p> <p>The <i>size</i> parameter specifies the number of array elements, as follows:</p> <ul style="list-style-type: none"> <li>0: 256</li> <li>1: 512</li> </ul> The default value is 0.

**SFTBLUT****ShiFT Binary LUT**

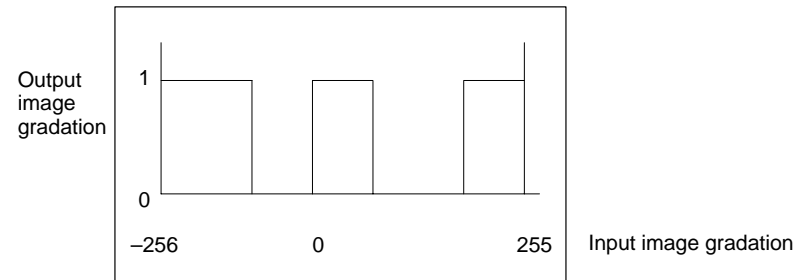
(Command)

<b>Action</b>	Shifts the binary LUT contents for each binary image plane.
<b>Format</b>	SFTBLUT <i>binary image plane#</i> , <i>shift</i>
<b>Example</b>	SFTBLUT 1,-12 Shifts the binary LUT for binary image plane# 1 by -12.
<b>Description</b>	The SFTBLUT command shifts the contents of the binary LUT for the specified binary image plane# by the amount specified with the shift parameter. <p>The shift operation with positive and negative <i>shift</i> parameters is shown below.</p>

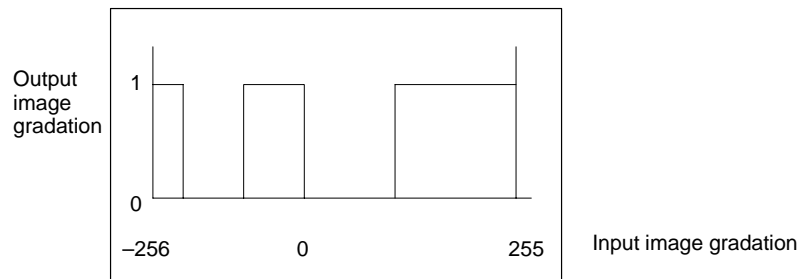
Before shifting:



Positive shift:



Negative shift:



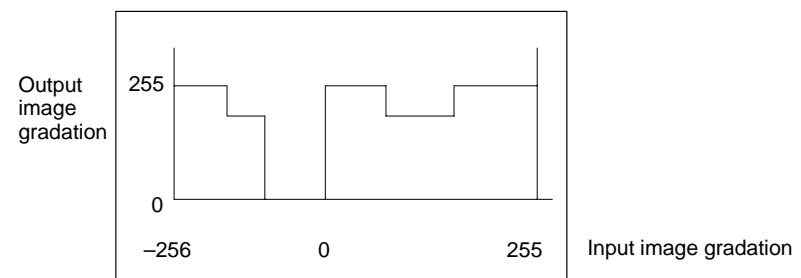
## SFTLUT

## SHIFT LUT

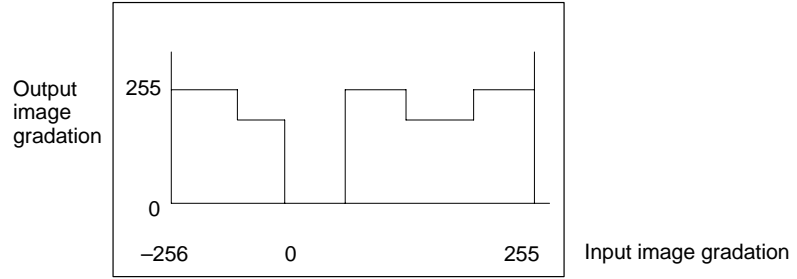
(Command)

<b>Action</b>	Shifts the filter LUT.
<b>Format</b>	<code>SFTLUT <i>shift</i></code>
<b>Example</b>	<code>SFTLUT -12</code> Shifts the filter LUT by $-12$ .
<b>Description</b>	The SFTLUT command shifts the contents of the filter LUT by the amount specified with the shift parameter. The shift operation with positive and negative <i>shift</i> parameters is shown below.

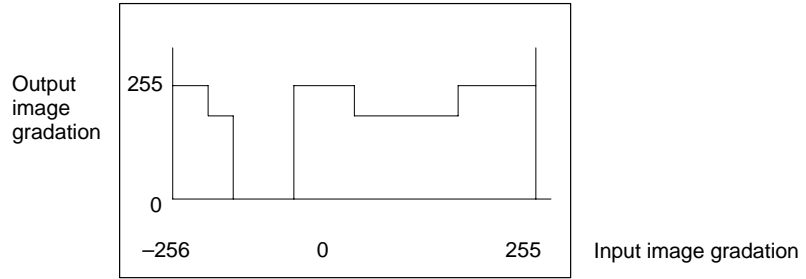
Before shifting:



Positive shift:



Negative shift:



## SGN

## SiGN

(Function)

- Action** Determines the positive or negative sign of a numeric expression.
- Format** `SGN (numeric expression)`
- Example** `A=SGN(-35)`  
Assigns -1 to variable A to indicate that the numeric expression -35 is negative.
- Description** The SGN function returns a value (-1, 0, 1) to indicate the sign of the numeric expression. The relationship between the sign of the *numeric expression* and returned value is shown below.

Numeric expression	Returned value
Positive	1
0	0
Negative	-1

## SIN

## SINe

(Function)

- Action** Determines the sine of a numeric expression.
- Format** `SIN (numeric expression)`
- Example** `A=SIN(30*3.14159/180)`  
Assigns the sine of 305 to variable A.
- Description** The SIN function returns a value between -1 and +1. The numeric expression must be set in radians. Convert an angle in degrees to radians by multiplying by  $\pi/180$ .  
  
The *numeric expression* may be in the form of an integer, long integer, or a single or double-precision real number. The SIN function always returns a double-precision real number.



## SPACE\$

## SPACE\$

(Function)

<b>Action</b>	Creates a character string comprised of the number of spaces specified by the numeric expression.
<b>Format</b>	SPACE\$ ( <i>numeric expression</i> )
<b>Example</b>	A\$=SPACE\$( 3 )  Creates a character string containing 3 space characters and assigns it to variable A\$.
<b>Description</b>	The SPACE\$ function returns a character string containing the number of space characters (CHR\$(32)) specified by the <i>numeric expression</i> .  Specify the <i>numeric expression</i> as a value between 0 and 255.  The STRING\$ function is similar to the SPACE\$ function. In the following examples, the STRING\$ and SPACE\$ functions return the same character string.  A\$=SPACE\$(10) .... assigns 10 spaces to variable A\$ A\$=STRING\$(10, 32) .... assigns 10 spaces (CHR\$(32)) to variable A\$

## SPC

## SPaCe

(Function)

<b>Action</b>	Outputs the specified number of spaces.
<b>Format</b>	SPC ( <i>numeric expression</i> )
<b>Example</b>	PRINT "OMRON" ; SPC(10) ; "F300"  "OMRON" and "F300" are displayed on the screen separated by 10 spaces.
<b>Description</b>	The SPC function creates a character string containing the number of spaces specified by the <i>numeric expression</i> . The SPC function cannot be used alone. Use it together with the PRINT command.  Specify the <i>numeric expression</i> as an integer between -32768 (-2 <sup>15</sup> ) and 32767 (2 <sup>15</sup> -1). A negative numeric expression (-2 <sup>15</sup> to -1) is treated as 0. If the designated characters do not fit in one line, the remainder obtained by dividing the number of the designated characters by the number of characters fitting in one line shall be used as the set value. An example is seen in the following.

← 512 characters →

```
PRINT "A" ; SPC ( 520 ) ; "B"
A_____B
```

```
PRINT "A" ; SPC ( 520 ) ; "B"
```

The number of characters in one line is 512.

$$520 / 512 = 1 \text{ (the remainder is 8)}$$

The screen output is as follows:

A\_\_\_\_\_B (8 spaces between A and B)

**SPCLOSE****Spline CLOSE**

(Command)

<b>Action</b>	Draws a region bounded by a spline curve in VRAM.
<b>Format</b>	SPCLOSE <i>number of data</i> , <i>X array</i> , <i>Y array</i> , <i>VRAM</i> , [, [ <i>page#</i> ] [, [ <i>density or drawing mode</i> ] [, <i>lineart</i> ]]]
<b>Example</b>	SPCLOSE 28,XD,YD,3,,128,0 Draws a region in image memory with drawing density 128 bounded by a spline curve defined by the 28 coordinate points contained in arrays XD and YD.
<b>Description</b>	The SPCLOSE command draws a region bounded by a spline curve. The spline curve is defined by the coordinate points in <i>X array</i> and <i>Y array</i> between the first array element and the element defined by the number of data parameter. The <i>number of data</i> must not exceed 64. Specify the <i>VRAM</i> where the region is drawn with a number, as follows: 0: Character memory 1: Graphic memory 2: Window memory 3: Image memory 4: Shading memory Omit the <i>page#</i> or set to 0. Specify the <i>drawing method</i> with the <i>drawing density</i> or the <i>drawing mode</i> . The default value is <i>drawing mode</i> , OR. Specify with the <i>lineart</i> parameter if the spline curve is an outline only or filled. 0: Filled spline curve 1: Spline curve outline only The default value is 0. When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

**SPLINE****SPLINE**

(Command)

<b>Action</b>	Draws a spline curve in VRAM.
<b>Format</b>	SPLINE <i>number of data</i> , <i>X array</i> , <i>Y array</i> , <i>VRAM</i> , [, [ <i>page#</i> ] [, [ <i>density or drawing mode</i> ]]]
<b>Example</b>	SPLINE 28,XD,YD,3,,128 Draws a spline curve in image memory defined by the 28 coordinate points contained in arrays XD and YD with drawing density 128.
<b>Description</b>	The SPLINE command draws a spline curve defined by the coordinate points in <i>X array</i> and <i>Y array</i> between the first array element and the element defined by the <i>number of data</i> parameter. The <i>number of data</i> must not exceed 64. Specify the <i>VRAM</i> where the spline curve is drawn with a number, as follows: 0: Character memory 1: Graphic memory 2: Window memory 3: Image memory 4: Shading memory Omit the <i>page#</i> or set to 0.

Specify the *drawing method* with the *drawing density* or the *drawing mode*. The default value is *drawing mode*, OR.

When writing to a frame memory, the contents of planes write protected with the MASKBIT command remain unchanged.

**SQR****Square Root**

(Function)

<b>Action</b>	Determines the square-root of a numeric expression.
<b>Format</b>	SQR ( <i>numeric expression</i> )
<b>Example</b>	A=SQR ( 100 ) Assigns 10, the square root of 100, to variable A.
<b>Description</b>	The SQR function determines the square-root of the specified <i>numeric expression</i> . The <i>numeric expression</i> must be 0 or a positive value.  The <i>numeric expression</i> may be in the form of an integer, long integer, or a single or double-precision real number. The SQR function always returns a double-precision real number.

**SSCROLL****Shading memory SCROLL**

(Command)

<b>Action</b>	Scrolls the shading master memory.
<b>Format</b>	SSCROLL X, Y[, <i>center of rotation X</i> , <i>center of rotation Y</i> , <i>angle of rotation</i> ]
<b>Example</b>	SSCROLL 50, 100, 256, 256, 30 Rotates the shading master memory 30% clockwise about the coordinates (256, 256) and scrolls it 50 pixels in the X direction and 100 pixels in the Y direction.
<b>Description</b>	The SSCROLL command scrolls the shading master memory in the X, Y, and $\theta$ directions.  Specify the amount of scroll in the X direction with the X parameter and the amount of scroll in the Y direction with the Y parameter. Specify the scroll amounts as a number of pixels between -1024 and +1023.  Specify the amount of rotational scroll in degrees with the <i>angle of rotation</i> parameter and the coordinates of the center of rotation with <i>center of rotation X</i> and <i>center of rotation Y</i> . No rotation occurs if these parameters are omitted. Specify the <i>center of rotation X</i> and <i>center of rotation Y</i> as a number of pixels between -1024 and +1023.  Scrolling is completed in the order: rotation, X, Y.

**STOP****STOP**

(Command)

<b>Action</b>	Stops program execution.
<b>Format</b>	STOP
<b>Example</b>	IF A\$="ST" THEN STOP Stops program execution if A\$ = "ST".

**Description** The STOP command stops program execution. It can be used anywhere inside the program.

**STOP ON/OFF/STOP****STOP key ON/OFF/STOP**

(Command)

**Action** Disables, enables, or stops interrupts from the STOP Key.

**Format** STOP ON *or* OFF *or* STOP

**Example** STOP OFF

Disables branching to the interrupt processing routine when the STOP Key is pressed.

**Description** The STOP command controls branching to an interrupt processing routine when the STOP Key is pressed.

**STOP ON:** The STOP ON statement enables the interrupt processing routine when the STOP Key is pressed. When the STOP Key is pressed, operation branches to the interrupt processing routine at the line# or label defined with the ON STOP GOSUB statement.

**STOP OFF:** The STOP OFF statement disables the interrupt processing routine when the STOP Key is pressed. When the STOP Key is pressed, program operation stops but does not branch to an interrupt processing routine.

**STOP STOP:** The STOP STOP statement stops interrupt processing when the STOP Key is pressed. When the STOP Key is pressed, operation does not immediately branch to the interrupt processing routine. However, immediately branching is enabled by the STOP ON statement, operation branches to the interrupt processing routine at the line# or label defined with the ON STOP GOSUB statement.

Note that if the STOP OFF/STOP command is used the STOP Key does not function as normal to stop program execution.

The interrupt subroutine must be defined with the ON STOP GOSUB statement before a STOP ON/OFF/STOP command is executed.

**STR\$****STRing\$**

(Function)

**Action** Converts a number to character string representation.

**Format** STR\$ (*numeric expression*)

**Example** A\$=STR\$(5692)

Converts the value 5692 to a character string and assigns it to variable A\$.

**Description** The STR\$ function converts the numeric expression to a character string. A *numeric expression* cannot be directly assigned to a character variable. First convert the *numeric expression* with the STR\$ function before assigning it to a character variable.

The first character in the character string is a space if the *numeric expression* is positive. The space is replaced by a minus sign if the numeric expression is negative.

The VAL function has the opposite action to the STR\$ function.

**STRCHK****STRobe ChEck**

(Function)

<b>Action</b>	Checks for incorrect strobe flashing.
<b>Format</b>	STRCHK
<b>Example</b>	D=STRCHK Assigns the result of the strobe check to variable D.
<b>Description</b>	The STRCHK function checks for incorrect strobe flashing. The function returns a number as follows: 0: Normal strobe flash non-0: No strobe flash or disconnected strobe cable. The bit position corresponds to the strobe number.

**STRING\$****STRING\$**

(Function)

<b>Action</b>	Creates a character string containing the specified number of the specified character.
<b>Format</b>	STRING\$ ( <i>number of characters, character string or numeric expression 2</i> )
<b>Example</b>	A\$=STRING\$(10, "*" ) Assigns a character string containing 10 asterisks to variable A\$.
<b>Description</b>	The STRING\$ function returns a character string containing the number of specified 1-byte characters specified by the numeric expression. 2-byte characters may not be specified. Specify the <i>number of characters</i> as a value between 1 and 255. Specify the character to fill the character string with the <i>character string or numeric expression 2</i> parameter. Specifying character string: Specify the character string as a single 1-byte character. Only the leading character is valid if more than one character string is specified. The STRING\$ function returns a character string filled with the leading character. Specifying numeric expression 2: Specify the numeric expression 2 as the decimal character code for a 1-byte character as an integer between 0 and 255. The SPACE\$ function is similar to the STRING\$ function. In the following examples, the STRING\$ and SPACE\$ functions return the same character string. A\$=STRING\$(10, 32) .... assigns 10 spaces (CHR\$(32)) to variable A\$ A\$=SPACE\$(10) .... assigns 10 spaces to variable A\$

**STRMODE****STRobe MODE**

(Command)

<b>Action</b>	Enables and disables strobe flashing on the FLASH command.
<b>Format</b>	STRMODE <i>strobe#, switch</i>
<b>Example</b>	STRMODE 1, 1 Strobe 1 flashes when the FLASH command is executed.

**Description** The STRMODE command enables and disables strobe flashing when the FLASH command is executed.

Specify the number of the strobe as a value between 0 and 7 with the *strobe#* parameter. Set *strobe#* to -1 to specify all the strobes. The specified strobe flashes when the FLASH command is executed if the *switch* parameter is set to 1. Set the switch parameter is set to 0 to disable the strobe flashing.

All strobes enabled with the STRMODE command flash simultaneously when the FLASH command is executed.

**SUB-END SUB**

**SUB-END SUB**

(Command)

**Action** Defines a structural subroutine called by the CALL command.

**Format** SUB *label* (*argument* [, *argument*...])  
*Statement in SUB block*  
 END SUB

**Example** SUB \*SUB1 (A\$,B%,C#)  
 B\$ = A\$ + "OMRON"  
 END SUB

Defines the label \*SUB1 as a structural subroutine.

**Description** The block between SUB-END SUB defines the structural subroutine called by the CALL command.

All variables used in a subroutine are treated as local variables. The variables specified by the *arguments* are assigned the values of the local variables after execution of the subroutine.

The *label* defines the name of the subroutine. This name is used to call the subroutine with the CALL command.

The *arguments* can be specified as any type of variables, except array variables. No logical limitation is placed on the number of arguments. However, the command line can physically accommodate up to 255 characters only.

No further SUB-END SUB command may be nested inside a subroutine block. Statements in subprogram blocks must be located after the main program.

**SWAP**

**SWAP**

(Command)

**Action** Switches two variables.

**Format** SWAP *variable 1*, *variable 2*

**Example** SWAP A#,B#

Switches the values of variables A# and B#.

**Description** The SWAP command switches the contents of *variable 1* and *variable 2*. Both variables must be of the same type.

**TAB**

**TABulate**

(Function)

**Action** Specifies the position to display characters.

**Format** TAB (*numeric expression*)

**Example** PRINT "1234567";TAB(10);"XYZ"

Displays "1234567" from the left-hand character position (character 0) and "XYZ" from the character 10 position.

**Description** The TAB function moves the cursor along a line by the specified number of characters from the left-hand position. If the character position specified by the *numeric expression* is less than the current cursor position, the cursor moves to the specified character position on the next line.

The TAB function cannot be used alone. Use it together with the PRINT command.

A negative *numeric expression* is treated as 0. If a positive *numeric expression* is specified which exceeds the number of character in a line, the number of characters in a line is subtracted from the specified *numeric expression* to determine the cursor position.

The differences between the TAB function and the similar SPC function are shown in the following table.

	TAB function	SPC function
Spaces	Outputs the number of spaces determined by subtracting the number of previously output characters from the numeric expression.	Outputs the number of spaces specified by the numeric-expression from the current cursor position.
Line feed	Cursor moves to the next line if the character position specified by the numeric expression is less than the current cursor position.	No line feed.

**TAN**

**TANgent**

(Function)

**Action** Determines the tangent of a numeric expression.

**Format** TAN (*numeric expression*)

**Example** A=TAN( 45\*3.14159/180)  
Assigns the tangent of 45% to variable A.

**Description** The TAN function returns an integer between -1.701411834604692D+38 and +1.701411834604692D+38. The numeric expression must be set in radians. Convert an angle in degrees to radians by multiplying by  $\pi/180$ .  
The *numeric expression* may be in the form of an integer, long integer, or a single or double-precision real number. The TAN function always returns a double-precision real number.

**TIME\$**

**TIME\$**

(Function, Command)

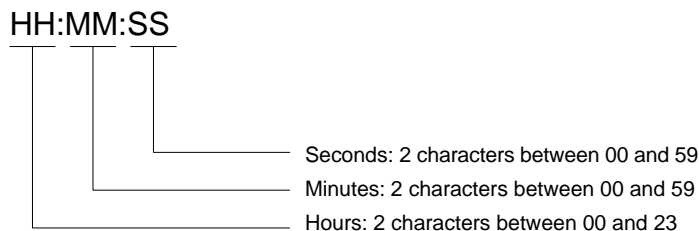
**Action** Displays and sets the time in the internal clock.

**Format** Format 1: TIME\$  
Format 2: TIME\$ = "HH:MM:SS"

**Example** CLOCK\$=TIME\$  
Assigns the time returned in TIME\$ to the character variable CLOCK\$.

**Description** Format 1  
Reads the time from the F300.

A character string in the format *HH:MM:SS* is returned when the TIME\$ function is executed.



The clock is set to 00:00:00 when the F300 power supply is turned on.

Format 2

- Sets the F300 clock:
- Set the hours between 00 and 23 with 2 characters.
- Set the minutes between 00 and 59 with 2 characters.
- Set the seconds between 00 and 59 with 2 characters.
- The time must be in the format HH:MM:SS. No part can be omitted.

**TIME\$ ON/OFF/STOP**

**TIME\$ ON/OFF/STOP**

(Command)

<b>Action</b>	Disables, enables, or stops timer interrupts.
<b>Format</b>	TIME\$ ON <i>or</i> OFF <i>or</i> STOP
<b>Example</b>	TIME\$ ON Enables branching to the interrupt processing routine due to the timer.
<b>Description</b>	The TIME\$ command controls branching to an interrupt processing routine due to the timer.
TIME\$ ON:	The TIME\$ ON statement enables the interrupt processing routine on timer operation. When timer set with the ON TIME\$ GOSUB command is reached, operation branches to the interrupt processing routine at the line# or label defined with the ON TIME\$ GOSUB statement.
TIME\$ OFF:	The TIME\$ OFF statement disables the interrupt processing routine on timer operation. All timer interrupts are ignored.
TIME\$ STOP:	The TIME\$ STOP statement stops interrupt processing on timer operation. When the set time is reached, operation does not immediately branch to the interrupt processing routine. However, immediately branching is enabled by the TIME\$ ON statement, operation branches to the interrupt processing routine at the line# or label defined with the ON TIME\$ GOSUB statement.

**TIMER**

**TIMER**

(Function, Command)

<b>Action</b>	Reads and sets the 10 ms timer.
<b>Format</b>	Format 1: TIMER Format 2: TIMER = <i>numeric expression</i>
<b>Example</b>	T&=TIMER Assigns the 10 ms timer PV to variable T&.
<b>Description</b>	<u>Format 1</u> Reads the 10 ms timer PV. The timer PV is 0 when the power is turned on and increases by 1 every 10 ms.



Format 2

Sets the 10 ms timer to the value between 0 and 2147483647. specified with the *numeric expression*.

**TROFF****TRace OFF**

(Command)

<b>Action</b>	Exits the Trace mode.
<b>Format</b>	TROFF
<b>Example</b>	TROFF Cancels the Trace mode.
<b>Description</b>	The TROFF command cancels the Trace mode. The Trace mode is cancelled by the TROFF command only. It is not cancelled when a program is executed, completed, interrupted, or re-executed.

**TRON****TRace ON**

(Command)

<b>Action</b>	Enters the Trace mode.
<b>Format</b>	TRON
<b>Example</b>	TRON Enters the Trace mode.
<b>Description</b>	The TRON command enters the Trace mode. When a program is executed in the Trace mode, the line numbers are displayed continuously on the screen as an aid to debugging. Displays from the program are unchanged in the Trace mode. However, program outputs become mixed with the line numbers. The Trace mode is cancelled by the TROFF command only. It is not cancelled when a program is executed, completed, interrupted, or re-executed.

**UBOUND****Upper BOUND**

(Function)

<b>Action</b>	Determines the upper boundary of an array dimension qualifier.
<b>Format</b>	UBOUND ( <i>array name</i> [, <i>number of dimensions</i> ])
<b>Example</b>	I=UBOUND ( A , 1 ) Assigns the upper limit of the qualifier of the 1-dimensional array A to variable I.
<b>Description</b>	The UBOUND function returns the upper boundary of an array dimension qualifier. Specify the name of the array for which the qualifier is to be determined with the <i>array name</i> parameter. Specify the number of dimensions of the array with the <i>number of dimensions</i> parameter. The default value is 1.

**UCASE\$****Upper CASE \$**

(Function)

<b>Action</b>	Converts lowercase letters in the character string to uppercase letters.
<b>Format</b>	UCASE\$ ( <i>character string</i> )
<b>Example</b>	A\$=UCASE\$ ( B\$ ) Converts all lowercase letters in the character string B\$ to uppercase letters and assigns the new character string to variable A\$.

**Description** The UCASE\$ function converts lowercase letters in the *character string* to uppercase letters. Existing uppercase letters remain unchanged.

**VAL**

**VALue**

(Function)

**Action** Converts a number represented as a character string to a number.

**Format** VAL (*character string*)

**Example** A=VAL( " 10 " )  
 Converts the character string "10" to the number 10 and assigns it to variable A.

**Description** The VAL function converts the character string to a real number.  
 The *character string* must be specified as a character variable or character constant starting with +, -, & or a digit between 0 and 9. The VAL function returns 0 if the *character string* does not conform to this format.  
 If the *character string* contains a character which cannot be converted to a number, the characters before the unconvertable character are converted. Spaces in the *character string* are ignored and the positions of the spaces are closed in the returned number.  
 The STR\$ function has the opposite action to the VAL function.

**VDWAIT**

**VD WAIT**

(Command)

**Action** Delays the VD interrupt the specified number of times.

**Format** VDWAIT *number of times*

**Example** VDWAIT 3  
 Delays the VD interrupt processing by 3 VD interrupts (approx. 50 ms).

**Description** The VDWAIT command counts the number of VD interrupts and delays VD interrupt processing the specified number of VD interrupts. Each VD interrupt is equivalent to 1/60 second (approx. 16.7 ms).  
 Program operation cannot be interrupted with the STOP Key or CNTRL+C Keys during VDWAIT operation.

**VIDEOIN**

**VIDEO IN**

(Command)

**Action** Inputs an image to the image memory.

**Format** VIDEOIN [*page#*] [, *input path*]

**Example** VIDEOIN 0,0  
 Input the image from image bus 1 to the image memory.

**Description** The VIDEOIN command reads image data to the image memory. This command is used to store an image before carrying out multiple measurements on the same image.  
 Omit the *page#* or set to 0.  
 Input the *path* from which the image is loaded with a number, as follows:  
 0: image bus 1  
 1: image bus 0  
 The default value is 0.

The image input timing in each input mode is shown below.

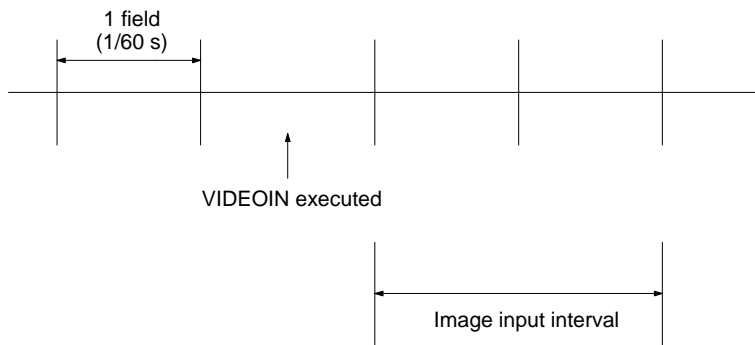


Image input from image bus 0 is disabled while the image memory contents are displayed.

## WDISP

## Window DISPLAY

(Command)

<b>Action</b>	Sets the type of image display in a window.
<b>Format</b>	WDISP <i>window plane#</i> , <i>image type</i> [, [ <i>display or not</i> ] [, <i>binary reverse</i> ]]
<b>Example</b>	WDISP 2, 1, 1 Display the binary image plane#2 window plane and set a binary image in the window.
<b>Description</b>	The WDISP command sets the type of image display inside the window plane specified by the <i>window plane#</i> . Set <i>window plane#</i> to -1 to specify all window planes.  Set the <i>image type</i> to 0 to display the same image type inside the window as outside the window. Specify 1 to display a binary image in the window.  Specify whether the window itself is displayed with the <i>display or not</i> item. Set to 1 to display the window or to 0 to hide the window. The default value is 0. If window display is selected, the window gradations are increased, or made lighter, to display the window.  Set the <i>binary reverse</i> parameter to 1 to reverse the displayed image or to 0 for a normal displayed image. The default value is 0. The <i>binary reverse</i> setting is valid only when the image type is set to 1 (binary image).

## WHILE-WEND

## WHILE-While END

(Command)

<b>Action</b>	The commands between the WHILE and WEND statements are executed repeatedly while a condition remains fulfilled.
<b>Format</b>	WHILE <i>logical expression</i> : : WEND
<b>Example</b>	100 WHILE KAI<10 : : 200 WEND  Executed the lines from 100 to 200 repeatedly while the value assigned to variable KAI is less than 10.

<b>Description</b>	<p>The commands between the WHILE and WEND statements (WHILE–WEND loop) are executed repeatedly while the logical expression remains true (non-0). The <i>logical expression</i> declared with the WHILE statement provides the condition for the commands to be executed. The WHILE–WEND loop is executed repeatedly while the logical expression remains true (non-0). Control is transferred to the line after the WEND statement when the logical expression becomes false (0).</p> <p>The WHILE–WEND loop is not executed if the logical expression is initially false (0). Control jumps immediately to the line after the WEND statement. Other WHILE–WEND loops may be nested inside a WHILE–WEND loop. The nested WHILE–WEND loops must have a one-to-one correspondence between the WHILE and WHEN statements from the inside of the nested loops toward the outside. WEND statements must not be omitted.</p>
--------------------	--

**WINDOW****WINDOW**

(Command)

<b>Action</b>	Draws the window set for the specified scene to the window memory.
<b>Format</b>	WINDOW <i>scene#</i>
<b>Example</b>	<p>WINDOW 3</p> <p>Draws the window set for scene #3.</p>
<b>Description</b>	<p>The WINDOW command refers to the window data previously set in the Menu mode. It draws the window data for all windows for the scene (0 to 15) specified with the <i>scene#</i> parameter. The previous window memory contents are deleted.</p> <p>To draw a complex window, it is convenient to set the window while looking at the workpiece in the Menu mode before executing the WINDOW command.</p>

**WRITE****WRITE**

(Command)

<b>Action</b>	Displays data on the display.
<b>Format</b>	WRITE <i>expression list</i> [{, <b>or</b> ;} <i>expression</i> ]
<b>Example</b>	<p>WRITE A\$, B</p> <p>Writes the data defined in variables A\$ and B to the text display.</p>
<b>Description</b>	<p>Displays the numeric data and character data specified with the <i>expression list</i> on the text display.</p> <p>List the numeric and character expressions in the <i>expression list</i> delimited by commas (,) or semicolons (;). Commas (,) or semicolons (;) are identical in function. If the expressions in the <i>expression list</i> are delimited by commas (,), the data items are displayed on the screen separated by commas (,). A character string is displayed enclosed in double-quotations (" ").</p> <p>If a numeric expression is specified, spaces in front of the data are deleted on the display.</p> <p>A carriage return character is inserted automatically after all the expressions are displayed.</p> <p>The PRINT command is similar to the WRITE command but differs as follows:</p> <ul style="list-style-type: none"> <li>• comma (,) and semicolon (;) delimiter characters differ in function</li> <li>• no commas (,) displayed between data items on the screen</li> <li>• double-quotations (") not displayed</li> </ul>

- spaces in front of numeric data are displayed.

**WRITE #****WRITE #**

(Command)

<b>Action</b>	Writes data to a sequential file.
<b>Format</b>	<code>WRITE # file#, expression list [{, or ;} expression]</code>
<b>Example</b>	<p><code>WRITE #1 , J , K , A\$</code></p> <p>Concatenates the numeric data and character data defined with the expressions J (numeric), K (numeric), and A\$ (character) and writes it to file #1.</p>
<b>Description</b>	<p>Specify the <i>file#</i> as the number in which the output file was opened with the OPEN command. Specify the same <i>file#</i> with the CLOSE command to close the file after output is complete.</p> <p>List the numeric and character expressions in the <i>expression list</i> delimited by commas (,) or semicolons (;). Commas (,) or semicolons (;) are identical in function. If the expressions in the <i>expression list</i> are delimited by commas (,), the data items are written to the file separated by commas (,).</p> <p>A character string is automatically enclosed in double-quotations (") when written to a file.</p> <p>If a numeric expression is specified, spaces in front of the data are deleted when the data is written to the file. Consequently, the file area used is less than with the PRINT# command which writes leading spaces to the file.</p> <p>The WRITE# command automatically inserts a linefeed character (CHR\$(10)) after writing the last expression to the file.</p> <p>The PRINT# command is similar to the WRITE# command but differs as follows:</p> <ul style="list-style-type: none"> <li>• comma (,) and semicolon (;) delimiters differ in function.</li> <li>• commas (,) and double quotations (") are not automatically inserted.</li> <li>• spaces in front of numeric data are also written to a file.</li> </ul>

**WSCROLL****Window SCROLL**

(Command)

<b>Action</b>	Scrolls the window memory.
<b>Format</b>	<code>WSCROLL X, Y[, center of rotation X, center of rotation Y, angle of rotation]</code>
<b>Example</b>	<p><code>WSCROLL 50 , 100 , 256 , 256 , 30</code></p> <p>Rotates the window memory 30% clockwise about the coordinates (256, 256) and scrolls it 50 pixels in the X direction and 100 pixels in the Y direction.</p>
<b>Description</b>	<p>The WSCROLL command scrolls the window memory in the X, Y, and <math>\theta</math> directions.</p> <p>Specify the amount of scroll in the X direction with the X parameter and the amount of scroll in the Y direction with the Y parameter. Specify the scroll amounts as a number of pixels between -1023 and +1023.</p> <p>Specify the amount of rotational scroll in degrees with the <i>angle of rotation</i> parameter and the coordinates of the <i>center of rotation</i> with <i>center of rotation X</i> and <i>center of rotation Y</i>. No rotation occurs if these parameters are omitted. Specify the <i>center of rotation X</i> and <i>center of rotation Y</i> as a number of pixels between -1023 and +1023.</p> <p>Scrolling is completed in the order: rotation, X, Y.</p>

# PART II

## Version 2.00

### SECTION 1

#### OVL Version 2.00 Improvements

This section describes the additional functions and improvements found in OVL Version 2.00.

1-1	Overview of Improvements	170
1-1-1	Summary of Additional OVL Capabilities	170
1-1-2	Alphabetical Listing of New Functions & Commands	171
1-1-3	Changes in Specifications	172
1-1-4	Increased Processing Speed	172
1-2	Additional OVL Capabilities	172
1-2-1	Multiple-window Functions	172
1-2-2	Window Enlargement/Reduction	174
1-2-3	Binary Image Processing	174
1-2-4	Raw Image Processing	175
1-2-5	Image-to-image Calculations	175
1-2-6	High-speed Array Operations	175
1-2-7	Renumbering Labels	176
1-2-8	Zooming Window & Shading Memory	176
1-2-9	Filtering Selection	177
1-2-10	Checking Menu Settings	177
1-2-11	Calculations using Simple Run Length	179
1-2-12	BCD/Binary Conversion	180
1-2-13	Console Key Interrupts	180
1-2-14	Memory Card Operations	180
1-2-15	Camera Synchronization	181
1-2-16	System Information	181
1-2-17	Standard I/O Settings	181
1-2-18	Loading/Saving Programs through RS-232C	181
1-2-19	Binary Level Setting Range	182

## 1-1 Overview of Improvements

### 1-1-1 Summary of Additional OVL Capabilities

The following capabilities have been added for Version 2.00. These improvements are described in detail in *PART II 1-2 Additional OVL Capabilities*.

<b>Multiple Window Functions</b>	The area, center of gravity, and axis angle can be calculated quickly with 9 or more windows.
<b>Window Enlargement/Reduction</b>	Window size can be enlarged and reduced to check for omission or blurring of characters.
<b>Binary Image Processing</b>	Binary image processing such as edge detection, changing line thickness, zooming, elimination of isolated points, filling in holes, and elimination of surrounding graphics can be performed.
<b>Raw Image Processing</b>	Raw image processing such as Sobel processing and 3×3 mask processing can be performed.
<b>Image-to-image Calculations</b>	Image-to-image calculations can be performed on binary and raw images.
<b>High-speed Array Operations</b>	A single command can be used to perform high-speed operations between array variables.
<b>Renumbering Labels</b>	Labels can be renumbered based on the windows' center of gravity or location on the screen.
<b>Zooming Window &amp; Shading Memory</b>	Window memory and shading memory can be enlarged and reduced. It is possible to switch quickly between several windows for measurements.
<b>Filtering Selection</b>	Previously, image filtering functions could only be set in the menu. These functions can now be selected in OVL as well.
<b>Obtain Menu Settings</b>	The criteria and reference image data set in the menu can now be obtained from OVL.
<b>Calculations using Simple Run Length</b>	A variety of measurements can be made based on the simple run length. This function can be used for quickly measuring dimensions, counting the number of IC pins, etc.
<b>BCD/Binary Conversion</b>	Two functions have been added to convert numerical values between BCD and binary. These functions are useful when outputting data from Parallel I/O Units or Terminal Block Units.
<b>Console Key Interrupts</b>	Several commands have been added to control Console key interrupts and define interrupt routines.
<b>Memory Card Operations</b>	The Memory Card can be formatted and the Memory Card's battery voltage can be checked from OVL.
<b>Camera Synchronization</b>	A command has been added that selects the method of camera synchronization (internal or external).
<b>System Information</b>	A new function has been added that displays the model and version of the system. This function can be used to ensure compatibility when writing programs.
<b>Standard I/O Settings</b>	Instructions can be input through the RS-232C port instead of a keyboard, and the video monitor output can be output to the RS-232C port, so an OVL program can be developed without a keyboard.
<b>Loading/Saving Programs through RS-232C</b>	The \$1A code at the end of files is now recognized when loading or saving programs through the RS-232C port using LOAD" COM:" or SAVE" COM:".
<b>Binary Level Setting Range</b>	The binary level range has been doubled, so the maximum binary level setting is now 511 rather than 255.

## 1-1-2 Alphabetical Listing of New Functions & Commands

The following table lists the functions and commands that have been added for OVL Version 2.00. (New features have been added to FILTDATA, IPL, LEVEL, and LSORT.)

Command/Function		Operation	Type	Page
A	ARRAYFUNC	Performs high-speed operations between the variables of up to 4 arrays.	Command	184
B	BATCHK	Checks the voltage of the Memory Card's battery.	Function	185
	BCDTOBIN	Converts BCD data to binary.	Function	185
	BEDGE	Detects edge of a binary image.	Command	185
	BINTOBCD	Converts binary data to BCD.	Function	186
	BMFUNC	Performs logic operations on binary images.	Command	186
C	CAMSYNC	Selects camera synchronization method.	Command	186
	CONKEY ON/ OFF/STOP	Enable and disable input of Console key interrupt. CONKEY STOP masks input of Console key interrupt. (Interrupt is recorded by not executed until CONKEY ON is executed.)	Command	187
D	DILA	Expands a binary image.	Command	187
E	EDGRJECT	Deletes the graphics surrounding a binary image.	Command	188
	ELIM	Eliminates isolated points from a binary image.	Command	188
	EROS	Reduces a binary image.	Command	189
F	FILTDATA	Specifies the line filter factors. (With Sobel processing)	Command	189
	FILTSEL	Selects image filtering functions.	Command	190
	FORMAT	Formats the Memory Card.	Command	191
G	GETMDATA	Obtains measurement data (MDATA) in a batch.	Command	191
	GETVER	Obtains the system's version.	Function	192
	GMFUNC	Performs logic operations on raw images.	Command	193
H	HFILL	Fills holes in binary images.	Command	193
I	IPL	Sets the OVL boot-up mode. (Loads and executes the designated file.)	Command	194
L	LEVEL	Sets the binary level for each binary image plane.	Command	195
	LSORT	Renumbers the assigned label numbers.	Command	195
M	MASK3	Performs 3×3 mask filtering on raw images.	Command	195
	MWMEAS	Performs multiple-window measurements.	Command	196
	MWSET	Defines windows for use in MWMEAS command.	Command	197
O	ON CONKEY GOSUB	Defines the interrupt subroutine that will be executed when a Console key interrupt input is received.	Command	197
R	RUNL2	Performs secondary processing at high speed on data obtained with the RUNL function.	Function	198
S	SCNJUDGE	Reads the criteria that were set in the menu mode.	Function	198
	SCNSTAND	Reads the reference image data that were set in menu mode.	Function	199
	SOBEL	Performs Sobel processing.	Command	199
	SZOOM	Zooms (reduces/enlarges) the shading memory.	Command	200
T	THIN	Reduces line thickness in a binary image.	Command	200
W	WDILA	Enlarges the binary image in window memory.	Command	200
	WEROS	Reduces the binary image in window memory.	Command	201
	WZOOM	Zooms (reduces/enlarges) the window memory.	Command	201



### 1-1-3 Changes in Specifications

The following changes have been made in OVL specifications.

#### File Delimiters

The following table shows the delimiters used with OVL Versions 1.□□ and 2.00. Relevant files are data files and programs transferred via RS-232C or between the CPU and a Memory Card.

File I/O	Delimiters used	
	Version 1.□□	Version 2.00
Input	CR+LF or LF	CR+LF or CR
Output	CR+LF	CR+LF

As shown in the table, Version 2.00 cannot handle version 1.□□ files that use LF as a delimiter. Be sure to change these delimiters to CR or CR+LF before inputting the files to OVL Version 2.00.

#### Compile Work Space

The factory setting for compile work space has been increased from 4K bytes to 8K bytes in order to prevent "Out of compile work space" errors. The maximum possible setting for compile work space is 16K bytes.

### 1-1-4 Increased Processing Speed

The following improvements have been made to increase processing speed.

#### Memory Card Access Speed

Data in Memory Cards is accessed 1.5 times faster (on average) when loading/saving programs or reading/writing data files. The increase in the access speed varies depending on the operation being performed.

#### Labelling Speed

The labelling operation performed with the LABEL command is 1.5 times faster (on average), although the increase in the labelling speed varies depending on the conditions.

The LDATA function obtains center-of-gravity data 1.5 times faster (on average), although any data less significant than 0.0001 is truncated.

## 1-2 Additional OVL Capabilities

This section describes the additional capabilities of OVL Version 2.00. Refer to *PART II 1-1-1 Summary of Additional OVL Capabilities* for a summary of these improvements.

### 1-2-1 Multiple-window Functions

This section describes how to use the new multiple-window functions added for OVL Version 2.00.

#### Multiple-window Features

The multiple-window functions allow high-speed measurements to be made with 9 or more windows. The basic features of multiple-window functions are listed below:

- 1, 2, 3... 1. The possible number of windows is limited only by memory capacity.
2. Arbitrary window graphics are possible.
3. The items that can be measured are the area, center of gravity, and axis angle within individual windows.
4. The measurement results are brought together and stored in array variables at high speed.
5. X and Y position compensation functions are provided.
6. White or black pixels can be specified for measurements.
7. Binary image planes can be specified for measurements.
8. The X center of gravity and Y center of gravity can be measured simultaneously.

**Multiple-window Applications** The multiple-window functions are useful in the following kinds of situations:

- There are a large number of relatively small measurement regions.
- Position compensation is performed in a large window, but the measurement region is small after position compensation.
- The presence of an object is measured by a large number of points.

**Using Multiple Windows** The basic procedure for using multiple-window functions is listed below:

- 1, 2, 3...**
1. If measurements will be made in a window that is not rectangular, the window graphic must be drawn in the window memory beforehand. It is not necessary to draw rectangular windows in window memory.
  2. Before beginning the measurement, set the binary image plane and measurement region (window) to be used in the measurement. Once these settings have been made, they are usually valid thereafter.
  3. Input the binary image in image memory. The image is normally input through image bus 0.
  4. Make the multiple-window measurement on the binary image that was input in step 3.
  5. Process the results of the multiple-window measurement.

There are two ways to specify the measurement region (window):

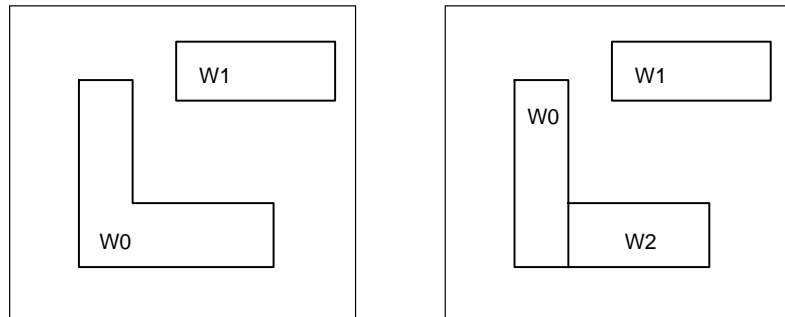
- The window's coordinates can be set one-by-one by the program. This method is convenient if the window coordinates are known beforehand.
- A window that has been set in menu mode can be registered as a measurement region. Use the labelling function and the window's coordinates will be calculated automatically by a program. A program that automatically calculates the coordinates is required, but this is a very easy and convenient way to set a window.

Refer to the sample programs in *PART II 3-1 Determination of Windows' ON/OFF Status* for a more detailed description of using multiple windows.

**Precautions for Multiple Windows**

Take the following points into consideration when using multiple-window functions:

- 1, 2, 3...**
1. The processing time is proportional to the size of the rectangle that encloses the measurement region (window).
  2. Measurements are made on the rectangle that encloses the measurement region. Consequently, if a second window is drawn within the rectangle that encloses an L-shaped window, the second window will be measured as the same window.



If W0 and W1 are set within the same window plane as shown in the diagram on the left, W1 will be within the rectangle that encloses W0. When W0 is measured, the processing will use W0 and part of W1 as the measurement region.

In these cases, split the window as shown in the diagram on the right. Splitting the window like this will increase the processing speed. Another solu-

tion would be to draw W0 and W1 in separate window planes and perform the multiple-window measurement twice.

- Multiple-window measurements cannot be performed in windows that have been enlarged or reduced with the WZOOM command, so do not use the multiple-window functions and WZOOM together.

## 1-2-2 Window Enlargement/Reduction

Windows can be enlarged with the WDILA command and reduced with the WEROS command. This enlargement and reduction is useful when checking for omission or blurring of characters.

```

10000 'For the label with the largest area, draw an
10010 'enlarged reverse window in window plane 7
10020 'and a reduced window in window plane 6.
10030 DISPLAY 31,0      'Binary image display
10040 WDISP -1,1,1     'Window display
10050 CLS 2
10060 RMODE 0,0,0      'Measure black pixels
10070 MEASURE          'Measure
10080 LABEL           'Labelling
10090 IF LNUM<0 THEN 10040
10100 LSORT 0         'Sort by area
10110 'Find coordinates of the enclosing rectangle.
10120 X1=LDATA(1,7)-10:Y1=LDATA(1,8)-10
10130 X2=LDATA(1,9)+10:Y2=LDATA(1,10)+10
10140 IF X1<1 THEN X1=1
10150 IF Y1<1 THEN Y1=1
10160 IF X2>510 THEN X2=510
10170 IF Y2>510 THEN Y2=510
10180 LPUTIMG 1,2,0,7  'Draw in window memory
10190 WDILA 7,X1,Y1,X2,Y2,,3      'Expand 3 times
10200 MASKBIT 2,0,&H7F 'Mask planes other than plane 7.
10210 BOX X1,Y1,X2,Y2,2,0,XOR     'Reverse window memory.
10220 MASKBIT 2,0,0
10230 LPUTIMG 1,2,0,6  'Draw in window memory
10240 WEROS 6,X1,Y1,X2,Y2,,3      'Reduce 3 times.

```

## 1-2-3 Binary Image Processing

Various kinds of binary image processing can be performed on binary images stored in image memory.

```

10000 'Detect edge in plane 7 in image memory.
10010 DISPLAY 31,3      'Binary image: bus 1
10020 BACKDISP 1,7     'Binary display plane 7.
10030 LEVEL 7,100,255  'Binary level setting.
10040 VIDEOIN 0,1:VDWAIT 3      'Input image
10050 FILTERIN 1       'Display image memory contents
10060 BEDGE 0,7       'Edge detection for entire image

```

## 1-2-4 Raw Image Processing

The MASK3 command performs 3×3 mask processing on raw images stored in image memory.

The SOBEL command performs sobel processing on raw images stored in image memory.

The FILTDATA and FILTER commands can be used for filter processing, but the FILTER command processing is performed with hardware using the high-speed interlaced method. On the other hand, the MASK3 and SOBEL command processing is performed with the non-interlaced method.

```
10000 'Sobel processing on a rectangular region.
10010 DISPLAY 31      'Raw image display.
10020 FILTERIN 0     'Camera image display.
10030 VIDEOIN:VDWAIT 3 'Input image
10040 FILTERIN 1     'Display image memory contents
10050 SOBEL 0,100,100,300,300      'Sobel processing of rectangular region
```

## 1-2-5 Image-to-image Calculations

The BMFUNC command can perform calculations between binary images.

The GMFUNC command can perform calculations between raw images, but only with F300-C11E models equipped with 2 image memories.

```
10000 'Displays only the part change in 1 second.
10010 DISPLAY 31,3    'Binary image: bus 1
10020 BACKDISP 1,7   'Binary display plane 7.
10030 MASKBIT 3,0,&H7F 'Mask planes other than plane 7.
10040 VIDEOIN 0,1    'Input binary image.
10050 VDWAIT 60      'Wait 1 second.
10060 MASKBIT 3,0,&HBF 'Mask planes other than plane 6.
10070 VIDEOIN 0,1    'Input binary image.
10080 VDWAIT 3       'Wait for image to be input.
10090 FILTERIN 1     'Display image memory contents
10100 BMFUNC 2,0,6,2,0,7      'Use XOR to calculate parts that don't match.
```

## 1-2-6 High-speed Array Operations

The ARRYFUNC command can be used to perform a variety of array operations at high speed. Be sure to declare the array variables with the DIM command before using ARRYFUNC.

The following examples show 5 applications of the ARRYFUNC command.

### Example 1

The content of array variable A is copied to array variable B.

```
ARRYFUNC 12,100,B,A
```

### Example 2

The content of array variable A is divided by 100.

```
FOR I=0 TO 99
  B(I)=100
NEXT
ARRYFUNC 6,100,A,A,B
```

### Example 3

The area within each window is compared to the lower and upper limits and the result of the comparison is stored in array variable JG.

```
GETMDATA 0,AR
ARRYFUNC 10,8,JG,AR,LO,HI
```

**Example 4**

The ON/OFF status in each window is output to a Terminal Block Unit.

```
GETMDATA 0,AR
ARRAYFUNC 11,8,JG&,AR,LO,HI
DOUT JG&(0),0,32
```

**Example 5**

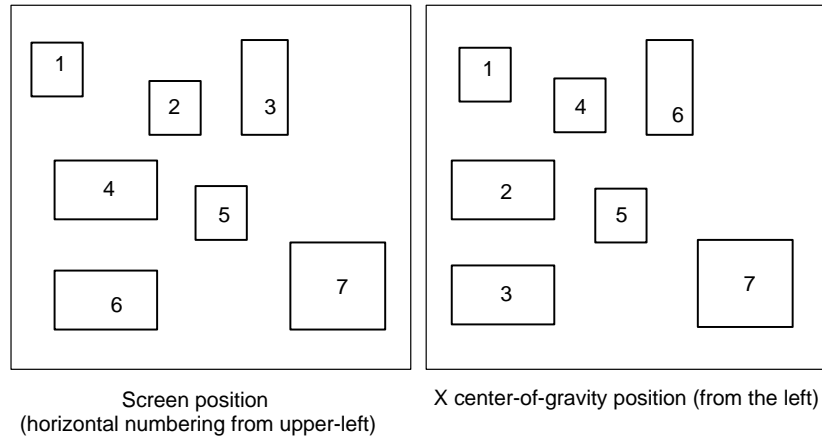
The ON/OFF status in 64 multiple windows is output to a Parallel I/O Unit.

```
MWMEAS 0,0,0,1,0,0,0,AR
ARRAYFUNC 11,64,JG&,AR,LO,HI
DOUT JG&(0),0,32
VDWAIT 1
DOUT JG&(1),0,32
```

**1-2-7 Renumbering Labels**

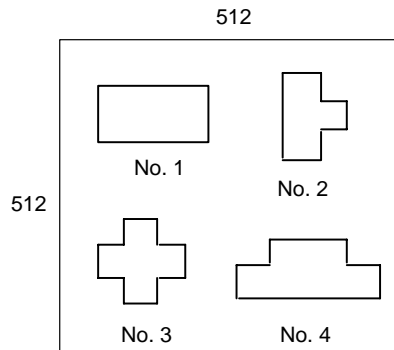
Previously, label numbers could be sorted according to their area only, but the LSORT command can renumber the labels according the windows' center of gravity or position on the screen.

The diagram on the left shows labelling according to screen position, beginning in the upper left corner. The diagram on the right shows labelling according to the location of the X center of gravity, beginning on the left. (The numbers indicate the renumbered labels.)



**1-2-8 Zooming Window & Shading Memory**

Window memory can be enlarged and reduced with the WZOOM command. Window memory can be used like a bank if the WZOOM and WSCROLL commands are used together.



When windows have been drawn as in the diagram above, the following commands would expand and measure window No. 1.

```
WZOOM 2
WSCROLL 0,0
MEASURE
```

The following commands would expand and measure window No. 3.

```
WZOOM 2
WSCROLL 0,-512
MEASURE
```

The coordinates specified by the WSCROLL command are performed on the coordinate space zoomed by the WZOOM command. If it is expanded by 2, the space would be (0,0) to (1023,1023). If it is reduced by 2, the space would be (0,0) to (255,255).

Operation of the SZOOM command is identical to that of the WZOOM command.

### 1-2-9 Filtering Selection

Image filtering functions can be selected easily with the FILTSEL command. Images are output from image bus 0 after image filtering.

When vertical edge or horizontal edge filtering are selected, the contents of the LUT (look-up table) for binary conversion are overwritten. With vertical edge or horizontal edge filtering it is necessary to set the binary conversion level for the negative portion of grayness.

Execute the following commands when setting the binary conversion level from L1 to L2:

```
LEVEL -1,L1,L2:LEVEL -1,512-L2,512-L1,OR
```

To display a raw image with strong smoothing:

```
DISPLAY 31,0
FILTSEL 2,0
```

To display a binary image with strong smoothing:

```
DISPLAY 31,0
FILTSEL 2,1
LEVEL -1,100,255 'Not necessary when specified beforehand.
```

To display a raw image with vertical edge filtering:

```
DISPLAY 31,0
FILTSEL 9,0 'Contents of the LUT for binary conversion are overwritten.
```

To display a binary image with vertical edge filtering:

```
DISPLAY 31,0
FILTSEL 9,1
LEVEL -1,100,512-100 'Absolutely necessary.
```

### 1-2-10 Checking Menu Settings

The criteria set with the menu can be read with the SCNJUDGE function.

This function is useful in a measurement program used for discrimination based on the criteria, when only the criteria are set in menu mode.

#### Example 1

```
10000 CHANGE 0 'Switch to scene 0.
10010 HI=SCNJUDGE(0,0,0,0) 'Upper limit for area of scene 0, window 0.
10020 LO=SCNJUDGE(0,0,0,1) 'Lower limit for area of scene 0, window 0.
```

```

10030 MEASURE          'Measure.
10040 A=MDATA(0,0)    'Obtain area.
10050 IF LO<=A AND A<=HI THEN PRINT "OK" ELSE PRINT "NG"
10060 GOTO 10030

```

**Example 2**

```

10000 'Display the criteria set with the menu.
10010 K$(0)="Area"
10020 K$(1)="X grav cent"
10030 K$(2)="Y grav cent"
10040 K$(3)="Axis angle"
10050 K$(4)="Edge angle"
10060 K$(5)="X center coor."
10070 K$(6)="Y center coor."
10080 K$(7)="Angle of inclination"
10090 K$(8)="Cross point X coordinate"
10100 K$(9)="Cross point Y coordinate"
10110 U$(0)="% "
10120 U$(1)="±PIX":U$(2)="±mm":U$(3)="±°"
10130 U$(4)="PIX":U$(5)="mm":U$(6)="°"
10200 FOR I=0 TO 9
10210   HI=SCNJUDGE(0,0,I,0)
10220   LO=SCNJUDGE(0,0,I,1)
10230   UN=SCNJUDGE(0,0,I,2)
10240   PRINT USING "@ UPPER LIMIT=#####.### @";K$(I),HI,U$(UN)
10250   PRINT USING "@ LOWER LIMIT=#####.### @";K$(I),LO,U$(UN)
10260 NEXT

```

The reference image data set with the menu can be read with the SCNSTAND function.

This function is useful when comparing the reference image data set with the menu or performing position compensation. All of the measurement item data will be renewed when reference image data is registered with the menu.

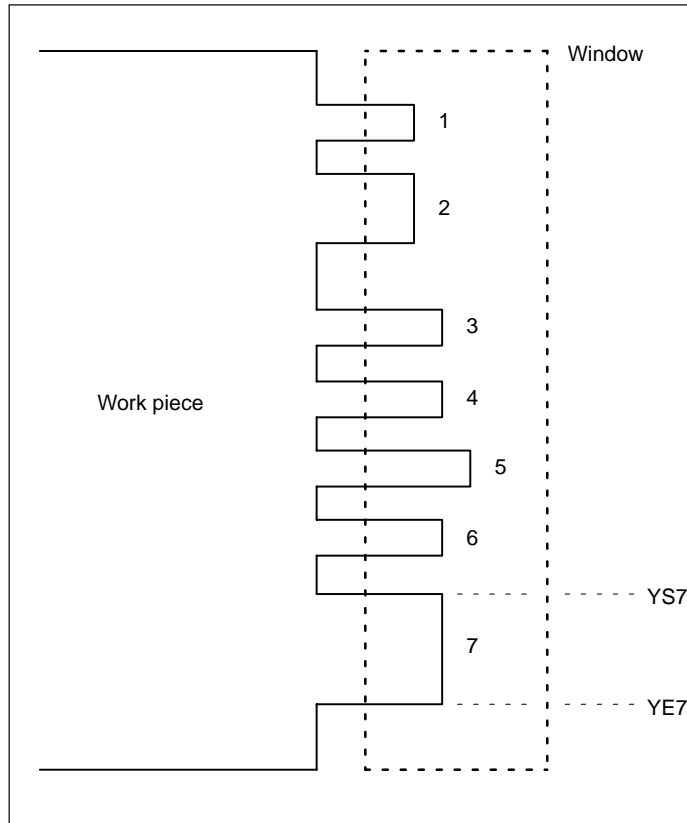
```

10000 'Display the reference image data set with the menu.
10010 DIM K$(11)
10020 K$(0)="Area reference"
10030 K$(1)="X grav cent reference"
10040 K$(2)="Y grav cent reference"
10050 K$(3)="Axis angle reference"
10060 K$(4)="Edge angle reference"
10070 K$(5)="X center coor. reference"
10080 K$(6)="Y center coor. reference"
10090 K$(7)="Angle of inclination reference"
10100 K$(8)="Cross point X coor. reference"
10110 K$(9)="Cross point Y coor. reference"
10120 K$(10)="X position compensation"
10130 K$(11)="Y position compensation"
10140 FOR I=0 TO 11
10150   PRINT USING "@ #####.###";K$(I),SCNSTAND(0,0,I)
10160 NEXT

```

### 1-2-11 Calculations using Simple Run Length

The RUNL2 function can be used to calculate a variety of values using the simple run length.



When the work piece and window are set up as shown in the diagram above, the various measurements described below can be made using RUNL2. The rectangular pieces projecting to the right from the work piece are known as legs. The Y coordinate of the upper edge of each leg is labelled YSn, and the Y coordinate of each lower edge is labelled YEn.

There are 17 types of measurements possible with RUNL2. These measurements, specified by numbers 0 to 16, are described below.

- 0: Area** The total area of the legs (area of the work piece within the window).
- 1: Number of Runs** The number of horizontal lines = the total height of the legs =  $\sum (YEn - YSn + 1)$ .
- 2: Average Run Length** The average length of all legs.
- 3: Maximum Run Length** The length of the longest leg (leg 5 in the diagram above).
- 4: Minimum Run Length** The length of the shortest leg (leg 1 in the diagram above).
- 5: Median Run Length** The length of the median leg (the middle of the group, ordered by length).
- 6: Max. Run Length Y Coord.** The Y coordinate of the longest leg. (YS5 in the diagram above)
- 7: Min. Run Length Y Coord.** The Y coordinate of the shortest leg. (YS1 in the diagram above)
- 8: Med. Run Length Y Coord.** The Y coordinate of the median leg.
- 9: Number of Run Groups** Number of legs.
- 10: Average Run Group Width** The average width of the legs.
- 11: Max. Run Group Width** The maximum value of leg width. (In the diagram above, leg 7's width =  $YE7 - YS1 + 1$ .)



<b>12: Min. Run Group Width</b>	The minimum value of leg width. (In the diagram above, leg 1's width = YE1–YS1+1.)
<b>13: Median Run Group Width</b>	The median leg width.
<b>14: Max. Run Group Width Y Coordinate</b>	The Y coordinate of the leg with the maximum width. (YS7 in the diagram above)
<b>15: Min. Run Group Width Y Coordinate</b>	The Y coordinate of the leg with the minimum width. (YS1 in the diagram above)
<b>16: Median Run Group Width Y Coordinate</b>	The Y coordinate of the leg with the median width.

## 1-2-12 BCD/Binary Conversion

The BCDTOBIN function converts BCD values to binary and the BINTOBCD function converts binary values to BCD. These functions are useful when BCD measurement results need to be output in binary. The BCD values must be 8 digits or less.

```
10000 CHANGE 0          'Scene 0 measurement conditions.
10010 MEASURE          'Measure.
10020 AREA=BINTOBCD(MDATA(0,0))      'Converts area to BCD.
10030 DOUT AREA,0,32   '8-digit BCD data is output in 32 bits.
```

## 1-2-13 Console Key Interrupts

The CONKEY ON, CONKEY OFF, and CONKEY STOP commands control Console key interrupts and the ON CONKEY GOSUB command defines the interrupt subroutine. Operation of the interrupt is identical to that of ON HELP GOSUB.

```
10000 ON CONKEY GOSUB *CONKEYINT
10010 CONKEY ON
10020 GOTO 10020
20000 *CONKEYINT
20010   PRINT "Console key interrupt occurred."
20020   PRINT "The key code was: ";HEX$(KEYIN(0))
20030 RETURN
```

## 1-2-14 Memory Card Operations

The BATCHK function is used to check the Memory Card's battery voltage. This is a convenient way to determine when the battery needs to be replaced.

```
SELECT BATCHK("C:")
  CASE 0:PRINT "Low voltage. Replace battery."
  CASE 1:PRINT "Insufficient voltage. Replace battery soon."
  CASE 1:PRINT "Battery voltage O.K."
END SELECT
```

The Memory Card can be formatted with the FORMAT command. All data (such as scene data saved in menu mode) will be erased when the Memory Card is formatted.

When FORMAT is executed, a prompt will appear to confirm that the Memory Card will be formatted. Input "Y" to proceed or "N" to cancel.

## 1-2-15 Camera Synchronization

The CAMSYNC command can be used to set the camera synchronization method to internal synchronization or external synchronization. When internal synchronization is selected, some time is required for the camera image to stabilize.

The selected synchronization method is valid for all cameras. It is not possible to set the synchronization of any particular camera or have some cameras with internal synchronization and some with external synchronization.

```
10000 'Switch to internal synchronization.
10010 CAMERA 0          'Select internal synchronization camera.
10020 CAMSYNC 1        'Select internal synchronization.
10030 VDWAIT 3         'Wait for image to stabilize.
```

## 1-2-16 System Information

The GETVER function can be used to display the OVL version and other system information. This function is useful when writing programs, since it can be used to ensure compatibility.

```
10000 IF GETVER(1)=11 THEN PRINT "There are 2 image memories."
10010 PRINT "OVL Version = ";GETVER(2)
10020 PRINT "System Version = ";GETVER(3)
10030 IF GETVER(2)<2.00 THEN
10040     PRINT "Multiple-window functions cannot be used."
10050 END IF
```

## 1-2-17 Standard I/O Settings

Instead of using a keyboard, commands can be input from a device connected to the RS-232C port. Instead of using a video monitor, data can be output to a device connected to the RS-232C port.

With standard I/O specified, settings can be made with IPL commands or the menu mode's "Y. System" "M.Initial Mode". The settings will be effective the next time that OVL is booted up, and will continue to be effective even after the power is turned off.

For a standard I/O device connected through the RS-232C port, channel 0 is fixed to 9600 baud, 8 data bits, no parity, and 1 stop bit. Screen control, such as clearing the screen, cannot be performed just by outputting characters through the RS-232C.

RS-232C file control cannot be performed on channel 0 when the RS-232C port is specified for standard I/O device.

## 1-2-18 Loading/Saving Programs through RS-232C

LOAD" COM:" and SAVE" COM:" can be used to load and save programs through the RS-232C port. The communications protocol is 9600 baud, 8 data bits, no parity, and 1 stop bit.

Programs are input through the RS-232C port with LOAD" COM:". The LOAD command ends when the file's end code (1A\$ in hexadecimal) is received. Programs are output through the RS-232C port with SAVE" COM:".

The following commands can be used to load and save screen data and image data through the RS-232C port. They are used just like LOAD" COM:" and SAVE" COM:".

SCNLOAD "COM:"	Loads screen data.
SCNSAVE "COM:"	Saves screen data.
IMGLOAD "COM:"	Loads image data.
IMGSAVE "COM:"	Saves image data.

## **1-2-19 Binary Level Setting Range**

In previous versions of OVL, the LEVEL command's binary level range was 0 to 255. In Version 2.00, this range has been doubled to 0 to 511. When filtering has been performed with the FILTDATA and FILTER commands, these values are used when pixel values (grayness values) are negative. The upper and lower limits 256 to 511 set with the LEVEL command correspond to pixel values (grayness values) of -256 to -1.

## **SECTION 2**

### **Reference**

This section provides detailed information on the new Version 2.00 commands and functions. Examples are also provided.

**ARRAYFUNC**

**ARRaY FUNction**

(Command)

<b>Action</b>	Performs a variety of array operations at high speed.
<b>Format</b>	ARRAYFUNC <i>operation, data elements, array name 1, array name 2</i> [, <i>array name 3</i> [, <i>array name 4</i> ]]
<b>Example</b>	ARRAYFUNC 10, 8, JUDGE, AREA, LO, HI
<b>Description</b>	The ARRAYFUNC command performs an operation between the variables of 2 to 4 arrays at one time, speeding up processing by eliminating the need for FOR-NEXT or other loop processing. For example, this function is useful when comparing measurement results to upper and lower limit values.

Specify the *data elements* as the number of array variables that will be used in the operation. The number of array variables must be declared beforehand using the DIM command, and the number of array variables must be greater than the specified number of *data elements*.

It is not necessary to include qualifiers or parentheses in the array name. When decimal points aren't used, the processing speed can be increased by using integer type arrays or long integer type arrays.

Specify the *operation* with one of the values listed below to determine the type of operation that will be performed. (A1 refers to array name 1, A2 refers to array name 2, A3 refers to array name 3, and A4 refers to array name 4.)

- 0: A1 = A2 AND A3
- 1: A1 = A2 OR A3
- 2: A1 = A2 XOR A3
- 3: A1 = A2 + A3
- 4: A1 = A2 - A3
- 5: A1 = A2 × A3
- 6: A1 = A2 ÷ A3
- 7: A1 = -1 (A2 = A3)  
= 0 (A2 <> A3)
- 8: A1 = -1 (A2 ≥ A3)  
= 0 (A2 < A3)
- 9: A1 = -1 (A2 ≤ A3)  
= 0 (A2 > A3)
- 10: A1 = -1 ((A3 ≤ A2) AND (A2 ≤ A4))  
= 0 ((A2 < A3) OR (A4 < A2))
- 11: The results of operation 10 (above) are substituted for A1 as a bit string. (This operation is described in more detail below.)
- 12: A1 = A2

- Note**
- 1. If A3 is 0 in the division in operation 6, the result will be 0.
  - 2. Overflow processing is not performed.

**Details of Operation 11**

The right sides (comparison results of A2, A3, and A4) are stored in the left side's array variable beginning with the least significant bit or in numerical order beginning with the one with the smallest element number.

When the right side's operation result is true, the bit value is 1. When the operation result is false, the bit value is 0.

When the left side's array variable is the integer type, the results of the right side's element numbers 0 to 15 are stored in element number 0, and the results of element numbers 16 to 31 are stored in element number 1. Thereafter the process is repeated.

When the left side's array variable is the long integer type, the results of the right side's element numbers 0 to 31 are stored in element number 0. Thereafter the process is repeated.

If this function is used, each window's area discrimination result is stored as bit information, so it is easy to determine each windows' OK/ON discrimination by outputting this bit information through an output port.

**BATCHK**

**BATtery Check**

(Function)

<b>Action</b>	Checks the voltage of the Memory Card's battery.
<b>Format</b>	BATCHK ( <i>drive</i> )
<b>Example</b>	BAT=BATCHK ( "C : " )
<b>Description</b>	Checks the voltage of the Memory Card's battery. One of the values listed below will be returned to indicate the battery status. 0: Low voltage (Battery must be replaced.) 1: Insufficient voltage (Battery should be replaced soon.) 2: Normal voltage (Battery is O.K.) The Memory Card is the "C" drive. An error will occur if a drive other than "C" is specified.

**BCD TO BIN**

**BCD TO BINary**

(Function)

<b>Action</b>	Converts BCD data to binary.
<b>Format</b>	BCD TO BIN ( <i>data</i> )
<b>Example</b>	A=BCD TO BIN (&H123)
<b>Description</b>	Converts BCD data to binary. BCD values between 0 and &H99999999 can be specified. This function is useful when inputting BCD data that was input to Terminal Block Units or Parallel I/O Units.

**BEDGE**

**Binary EDGE**

(Command)

<b>Action</b>	Detects edges of a binary image.
<b>Format</b>	BEDGE [ <i>page#</i> ], <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, <i>link evaluation constant</i> ]]]]]
<b>Example</b>	BEDGE 0, 7, 0, 0, 200, 200 BEDGE 0, 7
<b>Description</b>	Extracts edges from the binary image in image memory within the region defined by (X1, Y1) and (X2, Y2). This command is useful for detecting the outline of a work piece. The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0. The <i>binary image plane#</i> specifies the binary image plane in which edges will be detected. The start point ( <i>X1</i> , <i>Y1</i> ) and end point ( <i>X2</i> , <i>Y2</i> ) of a rectangular region can be specified to reduce the search region. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510). The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i> =0, and 4 linkages for any setting besides 0. The default value is 0. In edge detection processing, pixels with a value of 1 that are adjacent to a pixel with a value of 0 are set aside, and the other pixels are set to 0.

**BINTOBCD****BINary TO BCD**

(Function)

<b>Action</b>	Converts binary data to BCD.
<b>Format</b>	BINTOBCD ( <i>data</i> )
<b>Example</b>	A=BINTOBCD(100)
<b>Description</b>	Converts binary data to BCD. This function is useful when outputting measurement data to Terminal Block Units or Parallel I/O Units. Data values between 0 and 99999999 can be specified.

**BMFUNC****Binary Memory image FUNction**

(Command)

<b>Action</b>	Performs logic operations on two binary images.
<b>Format</b>	BMFUNC <i>operation</i> , [ <i>page#1</i> ], <i>plane#1</i> , [ <i>page#2</i> ], <i>plane#2</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ]]]]]
<b>Example</b>	BMFUNC 0 , , 0 , , 7
<b>Description</b>	<p>BMFUNC performs a logic operation on two binary images in <i>plane#1</i> and <i>plane#2</i>. The result of the operation is written to page number 2, plane number 2 of image memory.</p> <p>Three operations (AND, OR, and XOR) are available. Specify the operation by setting the <i>operation</i> parameter to one of the values listed below.</p> <p>0: AND 1: OR 2: XOR</p> <p>Specify the desired image memory page numbers with <i>page#1</i> and <i>page#2</i>. The default value is 0 for both <i>page#1</i> and <i>page#2</i>.</p> <p>The logic operation will be performed within the rectangular region specified by start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>). The coordinates can be set in the range 0 to 511. The default points are (0, 0) and (511, 511).</p>

**CAMSYNC****CAMera SYNChronization**

(Command)

<b>Action</b>	Selects the method of camera synchronization.
<b>Format</b>	CAMSYNC <i>sync mode</i>
<b>Example</b>	CAMSYNC 0
<b>Description</b>	<p>Set the <i>synchronization mode</i> to 0 to specify external synchronization, set a value other than 0 to specify internal synchronization.</p> <p>The selected synchronization method is valid for all cameras. It is not possible to set the synchronization mode of particular cameras.</p>

**CONKEY ON/OFF/STOP      CONsole KEY ON/OFF/STOP**

(Command)

<b>Action</b>	The CONKEY ON command enables Console key interrupts, the CONKEY OFF command disables Console key interrupts, and the CONKEY STOP command masks Console Key interrupts.
<b>Format</b>	CONKEY ON CONKEY OFF CONKEY STOP
<b>Example</b>	CONKEY ON
<b>Description</b>	<p>The CONKEY ON command enables Console key interrupts. When Console key interrupts are enabled, pressing the Console key will interrupt the main program and execute the interrupt subroutine defined by the ON CONKEY GOSUB command. CONKEY OFF disables Console key interrupts and CONKEY STOP masks Console Key interrupts.</p> <p>The following commands are related to CONKEY ON, CONKEY OFF, and CONKEY STOP:</p> <p style="text-align: center;">ON CONKEY GOSUB</p>

**DILA****DILAt**

(Command)

<b>Action</b>	Expands a binary image.
<b>Format</b>	DILA [ <i>page#</i> ], [ <i>binary image plane#</i> ] [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, [ <i>link evaluation constant</i> ] [, [ <i>enlargements</i> ]]]]]]]]
<b>Example</b>	DILA 0,7,0,0,200,200 DILA 0,7
<b>Description</b>	<p>Enlarges the binary image in image memory within the region defined by (X1, Y1) and (X2, Y2).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p> <p>The <i>binary image plane#</i> specifies the binary image plane in which the binary image will be expanded.</p> <p>The start point (X1, Y1) and end point (X2, Y2) can be input to specify a rectangular region that will be expanded. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510).</p> <p>The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i>=0, and 4 linkages for any setting besides 0. The default value is 0.</p> <p>The <i>enlargements</i> setting specifies the number of times that the image will be enlarged. The default value is 1.</p> <p>When a pixel within the rectangular region has a value of 0 and the adjacent pixel has a value of 1, enlargement processing will replace it with a value of 1.</p> <p>The specified rectangular region cannot be exceeded in enlargement. Enlargement processing will end before the enlargements setting is reached if it is not possible to enlarge the image further.</p>



**EDGRJECT****EDGE ReJECT**

(Command)

<b>Action</b>	Deletes exterior graphics from a binary image.
<b>Format</b>	EDGRJECT [ <i>page#</i> ], <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, <i>link evaluation constant</i> ]]]]]
<b>Example</b>	EDGRJECT 0,7,0,0,200,200 EDGRJECT 0,7
<b>Description</b>	<p>Deletes exterior graphics from the binary image in image memory within the region defined by (<i>X1</i>, <i>Y1</i>) and (<i>X2</i>, <i>Y2</i>).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p> <p>The <i>binary image plane#</i> specifies the binary image plane in which the exterior graphics will be deleted.</p> <p>The start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>) can be input to specify a rectangular region in which the exterior graphics will be deleted. The default points are (0, 0) and (511, 511).</p> <p>The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i>=0, and 4 linkages for any setting besides 0. The default value is 0.</p> <p>A "Vision error" will occur if the number of the diagram's borderlines exceeds 255 or the length of the diagram's borderline exceeds 4096.</p>

**ELIM****ELIMinate**

(Command)

<b>Action</b>	Eliminates isolated points from a binary image.
<b>Format</b>	ELIM [ <i>page#</i> ], <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, <i>link evaluation constant</i> ]]]]]
<b>Example</b>	ELIM 0,7,0,0,100,100 ELIM 0,7
<b>Description</b>	<p>Eliminates isolated points from a binary image in image memory within the region defined by (<i>X1</i>, <i>Y1</i>) and (<i>X2</i>, <i>Y2</i>).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p> <p>The <i>binary image plane#</i> specifies the binary image plane in which the isolated points will be eliminated.</p> <p>The start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>) can be input to specify a rectangular region the isolated points will be eliminated. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510).</p> <p>The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i>=0, and 4 linkages for any setting besides 0. The default value is 0.</p>

**EROS**

**EROSion**

(Command)

- Action** Reduces a binary image.
- Format** EROS [*page#*], *binary image plane#* [, [*X1*] [, [*Y1*] [, [*X2*] [, [*Y2*] [, [*link evaluation constant*] [, *reductions*]]]]]]]
- Example** EROS 0,7,0,0,200,200  
EROS 0,7
- Description** Reduces the binary image in image memory within the region defined by (X1, Y1) and (X2, Y2).  
The *page#* specifies the page number (0 or 1) in image memory. The default page number is 0.  
The *binary image plane#* specifies the binary image plane in which the binary image will be reduced.  
The start point (*X1*, *Y1*) and end point (*X2*, *Y2*) can be input to specify a rectangular region that will be reduced. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510).  
The *link evaluation constant* specifies the number of linkages that will be used in processing. There will be 8 linkages when the *link evaluation constant*=0, and 4 linkages for any setting besides 0. The default value is 0.  
The *reductions* setting specifies the number of times that the image will be reduced. The default value is 1.  
When a pixel within the rectangular region has a value of 1 and the adjacent pixel has a value of 0, reduction processing will replace it with a value of 0.  
The specified rectangular region cannot be exceeded in reduction. Reduction processing will end before the reductions setting is reached if it is not possible to reduce the image further.

**FILTDATA**

**FILTer DATA**

(Command)

- Action** Specifies the line filter factors.
- Format** FILTDATA *data 0*, *data 1*, ... , *data 8* [, [*global factor*] [, *sobel processing*]]
- Example** FILTDATA 0,1,1,1,1,1,1,1,1,8
- Description** The FILTDATA command sets the line filter factors. Either “2: sharpened image” or “5: line filter image” must be selected with the FILTER command to enable the line filter factors specified with the FILTDATA command.  
If “2: sharpened image” or “5: line filter image” have been selected with the FILTER command, filtering will be performed according to the following equation:  
$$\text{Pixel grayness} = (\sum \text{data } n \times M_n) \div (\text{global factor})$$
  
( $M_n$  indicates the grayness of the pixel itself and the adjacent pixels.)  
The line filter factors are specified with the parameters *data 0* to *data 8*. The positional relationship of these pixels is shown in the diagram below.

Data 1	Data 2	Data 3
Data 4	Data 0	Data 5
Data 6	Data 7	Data 8

The line filter factors for data 1 to data 8 can be set to 0,  $\pm 1$ ,  $\pm 2$ , or  $\pm 4$ . The line filter factor for data 0 can also be set to  $\pm 8$ .

The *global constant* parameter can be set to 0, 1, 2, 4, 8, or 16. The default value is 1.

Set the *sobel processing* parameter to 0 to disable sobel integration, set a value other than 0 to enable sobel integration. The default value is 0.

When sobel integration is enabled, the original image will undergo line filtering and then sobel processing. Sobel integration is effective only when "2: sharpened image" has been selected with the FILTER command.

## FILTSEL

## FILTer SElect

(Command)

<b>Action</b>	Specifies the type of image filtering.
<b>Format</b>	FILTSEL <i>type</i> [, <i>image type</i> ]
<b>Example</b>	FILTSEL 2
<b>Description</b>	The FILTSEL command specifies the type of image filtering. The image filtering specified with this command is the same as the filtering selected through the menu.

The *type* parameter specifies the type of filtering. Specify one of the values listed below:

- 0: OFF
- 1: Weak smoothing
- 2: Strong smoothing
- 3: Edge enhancement level 1
- 4: Edge enhancement level 2
- 5: Edge enhancement level 3
- 6: Edge enhancement level 4
- 7: Edge enhancement level 5
- 8: Relief
- 9: Vertical edges
- 10: Horizontal edges
- 11: All edges

When the FILTSEL command is executed, the information set with the FILTER and FILTDATA commands is overwritten and invalidated. When "9: Vertical edges" or "10: Horizontal edges" is selected, the binary level (contents of look-up table for binary conversion) is overwritten. The binary level must be set again.

The *image type* parameter determines whether the image is binary or raw. If necessary, specify the image type according to the current image display. (The default value is 0.)

- 0: Raw image (LUT OFF)
- 1: Binary image (LUT ON)

**FORMAT****FORMAT**

(Command)

<b>Action</b>	Formats a Memory Card.
<b>Format</b>	FORMAT <i>drive</i>
<b>Example</b>	FORMAT "C:"
<b>Description</b>	The FORMAT command formats a Memory Card. When the FORMAT command is executed, a prompt will appear (Are you sure? (Y/N)) to confirm that the Memory Card will be initialized. Press Y to proceed or N to cancel.

**GETMDATA****GET Measure DATA**

(Command)

<b>Action</b>	Obtains data generated by the MDATA function.
<b>Format</b>	GETMDATA <i>data type, array name</i> [, <i>subscript</i> ]
<b>Example</b>	GETMDATA 0,MD
<b>Description</b>	<p>The MDATA function obtains data one-by-one, and the GETMDATA command obtains this data in a batch of 8 windows and stores the data in array variables.</p> <p>The <i>data type</i> parameter specifies the type of data. Specify one of the values listed below:</p> <ul style="list-style-type: none"> <li>0: Area (pixel units)</li> <li>1: Area after calibration</li> <li>2: X center of gravity (pixel units)</li> <li>3: X center of gravity after calibration</li> <li>4: Y center of gravity (pixel units)</li> <li>5: Y center of gravity after calibration</li> <li>6: Axis angle (pixel units)</li> <li>7: Axis angle after calibration</li> <li>8: X center of gravity, Y center of gravity (pixel units)</li> <li>9: X center of gravity, Y center of gravity after calibration</li> <li>10: All (pixel units)</li> <li>11: All after calibration</li> </ul>

The *array name* parameter indicates the array variables in which the data will be stored. The number of array variables in the specified array must be declared beforehand using the DIM command. (It is not necessary to include qualifiers or parentheses in the array name.)

The number of array variables declared in DIM command must be greater than the number of data elements being obtained.

The first location in the array in which data will be stored can be specified in the *qualifier* parameter. The default location is the beginning of the array.

When the *data type* parameter is set to 0, data will be stored as follows:

```

A(0): Area of plane 0
A(1): Area of plane 1
      . . .
      . . .
      . . .
A(7): Area of plane 7

```

When the *data type* parameter is set to 8, data will be stored as follows:

- A(0): X center of gravity in plane 0
- A(1): X center of gravity in plane 1
- . . .
- . . .
- . . .
- A(7): X center of gravity in plane 7
- A(8): Y center of gravity in plane 0
- . . .
- . . .
- . . .
- A(15): Y center of gravity in plane 7

When the *data type* parameter is set to 10, data will be stored as follows:

- A(0): Area of plane 0
- A(1): Area of plane 1
- . . .
- . . .
- . . .
- A(7): Area of plane 7
- A(8): X center of gravity in plane 0
- . . .
- . . .
- . . .
- A(31): Axis angle in plane 7

**GETVER**

**GET VERSION**

(Function)

**Action**                      Obtains system version and other data.

**Format**                      GETVER (*type*)

**Example**                      MACHINE=GETVER ( 0 )

**Description**                The GETVER function can determine the system version and other data. This function can be used to create a single program that automatically compensates for hardware and software differences so that it can be run on different systems.

The *type* parameter specifies the type of data that will be obtained, as shown in the following table.

Type	Data obtained	Examples of returned values
0	Series-type	300 indicates the F300.
1	IMP Unit code	10: F300-C10EV2 11: F300-C11E
2	OVL version	Actual version number, such as 1.03 or 2.00
3	System version	Actual version number, such as 1.05 or 2.00

**GMFUNC** **Gray Memory image FUNction**

(Command)

<b>Action</b>	Performs operations on two raw images.
<b>Format</b>	GMFUNC <i>operation</i> , <i>page#1</i> , <i>page#2</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ]]]]]
<b>Example</b>	GMFUNC 7,0,1
<b>Description</b>	<p>GMFUNC performs an operation on two raw images within the rectangular region specified by start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>). The result of the operation is written to page number 2 of image memory.</p> <p>The following ten operations are available. Specify the operation by setting the <i>operation</i> parameter to the corresponding value from 0 to 9.</p> <ul style="list-style-type: none"> <li>0: AND</li> <li>1: OR</li> <li>2: XOR</li> <li>3: Addition 1 (Result set to 255 if greater than 255.)</li> <li>4: Addition 2 (Remainder of results divided by 256 if greater than 255.)</li> <li>5: Subtraction 1 (Result set to 0 if less than 0.)</li> <li>6: Subtraction 2 (Remainder of results divided by 256 if less than 0.)</li> <li>7: Maximum value</li> <li>8: Minimum value</li> <li>9: Absolute value (Absolute value of the difference of the two values.)</li> </ul>

Specify the desired image memory page numbers with *page#1* and *page#2*. The same page can be specified for both page numbers.

The operation will be performed within the rectangular region specified by start point (*X1*, *Y1*) and end point (*X2*, *Y2*). The coordinates can be set in the range 0 to 511. The default points are (0, 0) and (511, 511).

**HFILL****Hole FILL**

(Command)

<b>Action</b>	Fills holes in a binary image.
<b>Format</b>	HFILL [ <i>page#</i> ], <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, <i>link evaluation constant</i> ]]]]]
<b>Example</b>	HFILL 0,7,0,0,100,100 HFILL 0,7
<b>Description</b>	<p>The HFILL command fills holes in a binary image in image memory within the region defined by (<i>X1</i>, <i>Y1</i>) and (<i>X2</i>, <i>Y2</i>).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p> <p>The <i>binary image plane#</i> specifies the binary image plane in which the binary image will be processed.</p> <p>The start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>) can be input to specify a rectangular region that will be processed. The default points are (0, 0) and (511, 511).</p> <p>The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i>=0, and 4 linkages for any setting besides 0. The default value is 0.</p> <p>A "Vision error" will occur if the number of hole borderlines exceeds 255 or the length of the hole's borderline exceeds 4096.</p>

# IPL Initial Program Loading

(Command)

**Action** Sets the OVL boot-up mode. (Loads and executes the designated file.)

**Format** IPL [{0|1|file name}] [, [value 1] [, [value 2] [, [value 3] [, [value 4] [, [value 5] [, [standard input device] [, standard output device]]]]]]]]

**Example**

```
IPL 1
IPL "SAMPLE"
IPL , , , , , 1
```

**Description** The IPL command sets the OVL boot-up mode. The new settings are valid the next time that OVL is booted up, and the settings are retained even when the power is turned off.

The *0|1|file name* parameters specify the OVL boot-up procedure used when the power is turned on. (If a filename is specified, that file will be loaded and executed automatically.)

- 0: OVL will not be booted automatically.
- 1: The program in the IMP Unit will be booted automatically.

The *value 1* through *value 5* parameters specify various OVL settings, as shown below:

- Value 1: Number of files that can be open simultaneously. (1 to 11)
- Value 2: Array variable, character variable area (1K bytes to 62K bytes)
- Value 3: User stack area (1K bytes or 2K bytes)
- Value 4: Compiler area (2K bytes to 16K bytes)
- Value 5: Number of image display lines (20 or 25)

The *standard input device* parameter specifies the standard input device used when OVL has been started.

- 0: Keyboard
- 1: RS-232C

The *standard output device* parameter specifies the standard output device used when OVL has been started.

- 0: Video monitor
- 1: RS-232C

The settings will be displayed if no parameters are entered. The factory settings for these parameters are shown below:

Parameter	Use	Factory setting
Value 1	Number of files that can be open	11
Value 2	Array variable, character variable area	32 (K-bytes)
Value 3	User stack area	2 (K-bytes)
Value 4	Compiler area	8 (K-bytes)
Value 5	Number of image display lines	25
Standard input	Standard input device	0 (Keyboard)
Standard output	Standard output device	0 (Video monitor)

**LEVEL**

**LEVEL**

(Command)

<b>Action</b>	Sets the binary level for each binary image plane.
<b>Format</b>	LEVEL <i>binary image plane#</i> , <i>lower limit</i> , <i>upper limit</i> [, <i>mode</i> ]
<b>Example</b>	LEVEL -1, 128, 255 LEVEL 0, 0, 255, XOR
<b>Description</b>	<p>The LEVEL command sets the binary level for each binary image plane. Specify the binary image plane (0 to 7) for which the level is set with the <i>binary image plane#</i> parameter. Set the <i>binary image plane#</i> to -1 to set all binary image planes to the specified binary level.</p> <p>The <i>lower limit</i> and <i>upper limit</i> parameters specify the range for the binary level. Each parameter can be set from 0 to 511. When the level is within the range, the pixel is given a value of 1 (usually white).</p> <p>The <i>mode</i> parameter specifies the following operations. The default setting is "overwrite".</p> <ul style="list-style-type: none"> <li>OR: set the specified range to pixel value 1 (white).</li> <li>NOT: set the specified range to pixel value 0 (black).</li> <li>XOR: reverse the specified range</li> </ul>

**LSORT**

**Label SORT**

(Command)

<b>Action</b>	Renumbers the assigned label numbers.
<b>Format</b>	LSORT <i>mode</i>
<b>Example</b>	LSORT 0 LSORT 8
<b>Description</b>	<p>The LSORT command reorders and renumbers the labeled image data according to the specified ordering mode. The <i>mode</i> parameter specifies one of the 10 ordering modes listed below.</p> <ul style="list-style-type: none"> <li>0: Area, descending</li> <li>1: Area, ascending</li> <li>2: X center of gravity, descending</li> <li>3: X center of gravity, ascending</li> <li>4: Y center of gravity, descending</li> <li>5: Y center of gravity, ascending</li> <li>6: From upper left of screen, vertically</li> <li>7: From lower right of screen, vertically</li> <li>8: From upper left of screen, horizontally</li> <li>9: From lower right of screen, horizontally</li> </ul> <p>Be sure that the LABEL command has been executed before executing LSORT.</p>

**MASK3**

**MASK 3×3**

(Command)

<b>Action</b>	Performs 3×3 mask filtering on raw images.
<b>Format</b>	MASK3 [ <i>page#</i> ], <i>X1</i> , <i>Y1</i> , <i>X2</i> , <i>Y2</i> , <i>array name</i> [, <i>global factor</i> ]
<b>Example</b>	MASK3 0, 100, 100, 200, 200, TABLE, 2
<b>Description</b>	<p>The MASK3 command performs 3×3 mask filtering on raw images in image memory within the region defined by (X1, Y1) and (X2, Y2).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p>



The start point (*X1*, *Y1*) and end point (*X2*, *Y2*) can be input to specify a rectangular region that will be processed. The default points are (1, 1) and (511, 511). The coordinates can be set in the range 1 to 511.

The elements of the array can have values from -8 to +8. The following table shows the relationship between the array elements and the pixels.

TABLE (1)	TABLE (2)	TABLE (3)
TABLE (4)	TABLE (0)	TABLE (5)
TABLE (6)	TABLE (7)	TABLE (8)

The *global factor* parameter can be set to 0, 1, 2, 4, 8, or 16. The default value is 1.

## MWMEAS Multiple Window MEASure

(Command)

<b>Action</b>	Takes measurements of multiple windows.
<b>Format</b>	MWMEAS [ <i>page#</i> ], <i>binary image plane#</i> , <i>pixel type</i> [, [ <i>window operation</i> ] [, [ <i>X compensation</i> ] [, [ <i>Y compensation</i> ]]], <i>data type</i> , <i>data array</i>
<b>Example</b>	MWMEAS 0,0,1,1,0,0,0,MDAT
<b>Description</b>	Takes measurements of characteristics within windows registered with the MWSET command, and stores the results in array variables.

The *page#* specifies the page number (0 or 1) in image memory. The default page number is 0.

The *binary image plane#* specifies the binary image plane (0 to 7) in which the measurements will be taken.

The *pixel type* parameter specifies whether black or white pixels will be measured. Set this parameter to 0 to specify black pixels. A setting other than 0 specifies white pixels.

The *window operation* parameter determines whether the measurement will be made on the rectangular region defined in MWSET (*window operation*=0), or on the result of an AND between this region and the content of window memory (*window operation*=1).

The *X compensation* and *Y compensation* parameters can specify X and Y offsets from -512 to 511. The position compensation parameters cannot be used when the window operation parameter is set to 0.

The *data type* parameter specifies the type of data that will be measured. Specify one of the values listed below:

- 0: Area (pixel units)
- 1: Area after calibration
- 2: X center of gravity (pixel units)
- 3: X center of gravity after calibration
- 4: Y center of gravity (pixel units)
- 5: Y center of gravity after calibration
- 6: Axis angle (pixel units)
- 7: Axis angle after calibration
- 8: X, Y center of gravity (pixel units)
- 9: X, Y center of gravity after calibration

The *data array* parameter indicates the name of the array variables in which the data will be stored.

Data will be stored in the array in the order that the windows were registered with the MWSET command. When the *data type* parameter is set to X, Y center of gravity measurement, data will be stored as follows:

1<sup>st</sup> array element: X center of gravity for window 0  
 2<sup>nd</sup> array element: Y center of gravity for window 0  
 3<sup>rd</sup> array element: X center of gravity for window 1  
 4<sup>th</sup> array element: Y center of gravity for window 1

. . .  
 . . .  
 . . .

Array element 2n+1: X center of gravity for window n  
 Array element 2n+2: Y center of gravity for window n

**MWSET****Multiple Window SET**

(Command)

<b>Action</b>	Registers windows for multiple-window measurements.
<b>Format</b>	MWSET <i>binary image plane#, number of windows, X1 array, Y1 array, X2 array, Y2 array</i>
<b>Example</b>	MWSET 0,10,X1,Y1,X2,Y2
<b>Description</b>	<p>The MWSET command registers windows for multiple-window measurements by the MWMEAS command.</p> <p>The <i>binary image plane#</i> parameter specifies the binary image plane (0 to 7) in which the measurements will be taken.</p> <p>The <i>number of windows</i> parameter specifies the number of windows that will be registered.</p> <p>The <i>X1</i> and <i>Y1</i> arrays define the coordinates of upper left corners of the windows and the <i>X2</i> and <i>Y2</i> arrays define the coordinates of the lower right corners of the windows. The coordinates must be within the range 0 to 511.</p> <p>The number of array variables in the specified arrays must be declared beforehand using the DIM command. (It is not necessary to include qualifiers or parentheses in the array name.)</p> <p>The number of array variables declared in DIM must be greater than the number of windows that are being registered.</p>

**ON CONKEY GOSUB****ON CONSOLE KEY GOSUB**

(Command)

<b>Action</b>	Specifies the starting line number of the interrupt subroutine called by pressing the Console key.
<b>Format</b>	ON CONKEY GOSUB <i>line number or label</i>
<b>Example</b>	ON CONKEY GOSUB *CKEYINT
<b>Description</b>	<p>The ON CONKEY GOSUB command specifies the starting line number of the interrupt subroutine called by pressing the Console key. A label can be used instead of a line number to specify the start of the interrupt subroutine.</p> <p>The following commands are related to ON CONKEY GOSUB:</p> <p>CONKEY ON          CONKEY OFF          CONKEY STOP</p>

**RUNL2****RUN Length 2**

(Function)

<b>Action</b>	Performs high-speed processing on data obtained by the RUNL function.
<b>Format</b>	RUNL2 ( <i>binary image plane#, Y1, Y2, data measured</i> )
<b>Example</b>	A=RUNL2 (0,100,200,0)
<b>Description</b>	<p>The RUNL2 function processes the simple run length data to obtain more detailed information. In order to use the RUNL2 function, the run function must be turned on with the MMODE command and the simple run length data must be measured.</p> <p>The <i>Y1</i> and <i>Y2</i> parameters define the measurement range (on the Y-axis) and must be within the range 0 to 479.</p> <p>The <i>data measured</i> parameter specifies the type of data that will be measured. Specify one of the values listed below:</p> <ul style="list-style-type: none"> <li>0: Area</li> <li>1: Number of runs</li> <li>2: Average run length</li> <li>3: Maximum run length</li> <li>4: Minimum run length</li> <li>5: Median run length</li> <li>6: Y coordinate of maximum run length</li> <li>7: Y coordinate of minimum run length</li> <li>8: Y coordinate of median run length</li> <li>9: Number of run groups</li> <li>10: Average run group width</li> <li>11: Maximum run group width</li> <li>12: Minimum run group width</li> <li>13: Median run group width</li> <li>14: Y coordinate of maximum run group width</li> <li>15: Y coordinate of minimum run group width</li> <li>16: Y coordinate of median run group width</li> </ul>

**SCNJUDGE****SCeNe JUDGEment**

(Function)

<b>Action</b>	Obtains the criteria that were set in menu mode.
<b>Format</b>	SCNJUDGE ( <i>scene number, binary image plane#, item, data type</i> )
<b>Example</b>	LO=SCNJUDGE (0,0,0,1)
<b>Description</b>	<p>The SCNJUDGE function obtains the criteria for measurement items that were set in menu mode.</p> <p>The <i>scene number</i> parameter specifies the scene number (0 to 15).</p> <p>The <i>binary image plane#</i> parameter specifies the binary image plane (0 to 7).</p> <p>The <i>item</i> parameter specifies the measurement item. Specify one of the values listed below:</p> <ul style="list-style-type: none"> <li>0: Area</li> <li>1: X center of gravity</li> <li>2: Y center of gravity</li> <li>3: Axis angle</li> <li>4: Edge angle</li> <li>5: X center coordinate</li> <li>6: Y center coordinate</li> <li>7: Angle of inclination</li> <li>8: Cross point X coordinate</li> <li>9: Cross point Y coordinate</li> </ul>

The *data type* parameter specifies the type of data:

- 0: Upper limit
- 1: Lower limit
- 2: Unit

When "2: Unit" is specified for the data type, the SCNJUDGE function will return one of the following values:

- 0: % of reference image
- 1: Pixel difference to reference image (pixel units)
- 2: Pixel difference to reference image after calibration
- 3: Angle difference to reference image
- 4: Actual work measurement in pixels (pixel units)
- 5: Actual work measurement after calibration
- 6: Actual work measurement in degrees

## SCNSTAND

## SCeNe STANDard

(Function)

<b>Action</b>	Obtains the reference image data that were set in menu mode.
<b>Format</b>	SCNSTAND ( <i>scene number</i> , <i>binary image plane#</i> , <i>data type</i> )
<b>Example</b>	AR=SCNSTAND ( 0 , 0 , 0 )
<b>Description</b>	<p>The SCNSTAND function obtains the reference image data that were set in menu mode.</p> <p>The <i>scene number</i> parameter specifies the scene number (0 to 15).</p> <p>The <i>binary image plane#</i> parameter specifies the binary image plane (0 to 7).</p> <p>The <i>data type</i> parameter specifies the type of data. Specify one of the values listed below:</p> <ul style="list-style-type: none"> <li>0: Area</li> <li>1: X center of gravity</li> <li>2: Y center of gravity</li> <li>3: Main axis angle</li> <li>4: Edge angle</li> <li>5: X center coordinate</li> <li>6: Y center coordinate</li> <li>7: Angle of inclination</li> <li>8: X coordinate of cross point</li> <li>9: Y coordinate of cross point</li> <li>10: Reference X coordinate for X compensation</li> <li>11: Reference Y coordinate for Y compensation</li> </ul>

## SOBEL

## SOBEL

(Command)

<b>Action</b>	Performs sobel processing.
<b>Format</b>	SOBEL [[ <i>page#</i> ] [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, <i>Y2</i> ]]]]
<b>Example</b>	SOBEL , 100 , 100 , 200 , 200
<b>Description</b>	<p>The SOBEL command performs sobel processing within the rectangular region defined by (X1, Y1) and (X2, Y2).</p> <p>The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0.</p> <p>The start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>) can be input to specify a rectangular region that will be processed. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510).</p>

**SZOOM****Shading memory ZOOM**

(Command)

<b>Action</b>	Zooms (enlarges or reduces) the shading memory.
<b>Format</b>	SZOOM [ <i>zoom factor</i> [, <i>X reference coordinate</i> , <i>Y reference coordinate</i> ]]
<b>Example</b>	SZOOM 4
<b>Description</b>	The SZOOM command zooms (enlarges or reduces) the shading memory. The <i>zoom factor</i> parameter specifies the factor by which the shading memory will be magnified (0.25 to 512.00). The default zoom factor is 1. The X and Y reference coordinates define the point of reference for zooming. The default point is (0,0).

**THIN****THIN**

(Command)

<b>Action</b>	Reduces line thickness for a binary image in image memory.
<b>Format</b>	THIN [ <i>page#</i> ], <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, <i>link evaluation constant</i> ]]]]]]
<b>Example</b>	THIN 0,7,0,0,100,200 THIN 0,7
<b>Description</b>	The THIN command reduces line thickness for binary images in image memory within the rectangular region defined by (X1, Y1) and (X2, Y2). The <i>page#</i> specifies the page number (0 or 1) in image memory. The default page number is 0. The <i>binary image plane#</i> specifies the binary image plane in which the binary image will be processed. The start point (X1, Y1) and end point (X2, Y2) can be input to specify a rectangular region that will be processed. The coordinates can be set in the range 0 to 511. The default points are (0, 0) and (511, 511). The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i> =0, and 4 linkages for any setting besides 0. The default value is 0.

**WDILA****Window DILAtE**

(Command)

<b>Action</b>	Enlarges the binary image in window memory.
<b>Format</b>	WDILA <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, [ <i>link evaluation constant</i> ] [, <i>enlargements</i> ]]]]]]]]
<b>Example</b>	WDILA 7,0,0,100,100 WDILA 7,0
<b>Description</b>	The WDILA command enlarges the binary image in window memory within the region defined by (X1, Y1) and (X2, Y2). The <i>binary image plane#</i> specifies the binary image plane in which the binary image will be reduced. The start point (X1, Y1) and end point (X2, Y2) can be input to specify a rectangular region that will be enlarged. The default points are (1, 1) and (510, 510). The coordinates can be set in the range 1 to 510.

The *link evaluation constant* specifies the number of linkages that will be used in processing. There will be 8 linkages when the *link evaluation constant*=0, and 4 linkages for any setting besides 0. The default value is 0.

The *enlargements* setting specifies the number of times that the image will be enlarged. The default value is 1.

When a pixel within the rectangular region has a value of 0 and the adjacent pixel has a value of 1, enlargement processing will replace it with a value of 1.

The specified rectangular region cannot be exceeded in enlargement. Enlargement processing will end before the enlargements setting is reached if it is not possible to enlarge the image further.

**WEROS****Window EROSION**

(Command)

<b>Action</b>	Reduces the binary image in window memory.
<b>Format</b>	WEROS <i>binary image plane#</i> [, [ <i>X1</i> ] [, [ <i>Y1</i> ] [, [ <i>X2</i> ] [, [ <i>Y2</i> ] [, [ <i>link evaluation constant</i> ] [, <i>reductions</i> ]]]]]]]
<b>Example</b>	WEROS 7,0,0,100,100 WEROS 7,0
<b>Description</b>	<p>Reduces the binary image in window memory within the region defined by (<i>X1</i>, <i>Y1</i>) and (<i>X2</i>, <i>Y2</i>).</p> <p>The <i>binary image plane#</i> specifies the binary image plane in which the binary image will be reduced.</p> <p>The start point (<i>X1</i>, <i>Y1</i>) and end point (<i>X2</i>, <i>Y2</i>) can be input to specify a rectangular region that will be reduced. The coordinates can be set in the range 1 to 510. The default points are (1, 1) and (510, 510).</p> <p>The <i>link evaluation constant</i> specifies the number of linkages that will be used in processing. There will be 8 linkages when the <i>link evaluation constant</i>=0, and 4 linkages for any setting besides 0. The default value is 0.</p> <p>The <i>reductions</i> setting specifies the number of times that the image will be reduced. The default value is 1.</p> <p>When a pixel within the rectangular region has a value of 1 and the adjacent pixel has a value of 0, reduction processing will replace it with a value of 0.</p> <p>The specified rectangular region cannot be exceeded in reduction. Reduction processing will end before the reductions setting is reached if it is not possible to reduce the image further.</p>

**WZOOM****Window memory ZOOM**

(Command)

<b>Action</b>	Zooms (enlarges or reduces) the window memory.
<b>Format</b>	WZOOM [ <i>zoom factor</i> ] [, <i>X reference coordinate</i> , <i>Y reference coordinate</i> ]]
<b>Example</b>	WZOOM 4
<b>Description</b>	<p>The WZOOM command zooms (enlarges or reduces) the window memory.</p> <p>The <i>zoom factor</i> parameter specifies the factor by which the window memory will be magnified (0.25 to 512.00). The default zoom factor is 1.</p> <p>The <i>X</i> and <i>Y</i> reference coordinates define the point of reference for zooming. The default point is (0,0).</p>

When the WZOOM command is executed, WSCROLL 0,0 is executed internally. After the WZOOM command has been executed, the scroll quantities and rotation center specified with the WSCROLL command are specified for the coordinate space after zooming.

Measurements can be made on the zoomed window image (the image displayed on the screen).

# SECTION 3

## Sample Programs

This section provides sample programs using the new Version 2.00 commands and functions.

- 3-1 Determination of Multiple Windows' ON/OFF Status ..... 204
  - 3-1-1 Introduction ..... 204
  - 3-1-2 Program ..... 204
- 3-2 Shape Inspection using Window Enlargement/Reduction ..... 208
  - 3-2-1 Introduction ..... 208
  - 3-2-2 Program ..... 208



## 3-1 Determination of Multiple Windows' ON/OFF Status

### 3-1-1 Introduction

This program uses the multiple-window functions to determine the ON/OFF status of each window. Perform the operations below in menu mode beforehand.

- 1, 2, 3...**
1. Set up several windows for multiple-window use in window plane 0.
  2. Set up a window in window plane 7 for position compensation.
  3. Set the binary level.
  4. If necessary, register a reference image for position compensation.

Once the program has been executed, the ON/OFF status of each window can be determined using the following operations:

#### Measurement

Measure when the Console's Enter Key is pressed or the STEP signal goes ON. Measure the area of black pixels in the window. The percentage of black pixels should be greater than 50% in each window. The results are output to the screen and Parallel I/O Unit. The output to the Parallel I/O Unit will be ON if the result for that window is OK, and OFF if the result is not NG.

**Reference Image Registration** Register the reference image when the Console's Shift+Enter Keys are pressed or DI5 is ON and the STEP signal goes ON. Window plane 7's center of gravity coordinates are recorded as the reference position.

### 3-1-2 Program

```

10000 / *****
10010 /                               (c)Copyright OMRON Co. 1992
10020 /                               All Rights Reserved
10030 / -----
10040 /                               F300 Sample Program
10050 'Multiple-window sample program #1
10060 '  Determines whether the area in each window is OK/NG.
10070 '  It is OK if 50% or more of the area is black.
10080 '  Results output to the screen and Parallel I/O.
10090 'Set the following conditions in menu mode for scene 0:
10100 '  Window plane 0: Multiple windows
10110 '  Window plane 7: Window for position compensation
10120 'Procedure details
10130 / *****
10140 / =====
10150 'Data definitions
10160 / =====
10170 CLEAR
10180 W.SCN=0                               'Scene number
10190 W.MAX=255                             'Max. number of windows
10200 N=W.MAX
10210 DIM W.X1%(N),W.Y1%(N)                'Start points defining multiple windows
10220 DIM W.X2%(N),W.Y2%(N)                'End points defining multiple windows
10230 DIM W.WA%(N)                          'Areas of multiple windows
10240 DIM W.AR%(N)                          'Reference areas of multiple windows

```

```

10250 DIM W.LO%(N),W.HI%(N)      'Upper/lower limits for area discr.
10260 DIM W.JG&(N/32)          'Storage of discrimination results
10270 W.LO=50                  'Lower limit reference % for area discr.
10280 W.HI=100                 'Upper limit reference % for area discr.
10290 W.BW0=0                  'Black/white specification of plane 0
10300 W.BW7=0                  'Black/white specification of plane 7
10310 W.SX=0:W.SY=0           'Reference image X,Y
10320 W.DX=0:W.DY=0           'Compensation X,Y
10330 '-----
10340 'Main process
10350 '-----
10360 *MAIN
10370   GOSUB *HWINIT
10380   GOSUB *INITIAL
10390   GOSUB *MWINDSET
10400 ON INTR GOSUB *INTR1:INTR ON
10410 ON CONKEY GOSUB *INTR2:CONKEY ON
10420 WHILE (1)
10430 WEND
10440 '-----
10450 'Initialization process
10460 '-----
10470 *INITIAL
10480   CONSOLE ,,0:LOCATE ,,0
10490   DISPLAY 31              'Displays the camera image
10500   WINDOW W.SCN            'Window drawing
10510   CAMMODE 1               'Output of frame buffer contents
10520   FLASH 2                 'Shutter with STEP synchronization
10530   MMODE 7,W.BW7,1         'Measurement mode for plane 7
10540   W.SX=SCNSTAND(W.SCN,7,1) 'Obtain reference X center of gravity
10550   W.SY=SCNSTAND(W.SCN,7,2) 'Obtain reference Y center of gravity
10560 RETURN
10570 '-----
10580 'Hardware initialization
10590 '-----
10600 *HWINIT
10610   CAMERA 0:CAMMODE 0:FLASH 1:FILTERIN 0:FILTER 0,1,0
10620   MASKBIT 2,0,0:MASKBIT 3,0,0:MASKBIT 4,0,0
10630   CLS 0:CLS 1:CLS 2:CLS 3:SBANK 1:CLS 4:SBANK 0:CLS 4
10640   LEVEL -1,128,255:WSCROLL 0,0:SSCROLL 0,0:WZOOM:SZOOM
10650   DIM LUTDATA0%(256),LUTDATA1%(256)
10660   FOR I=0 TO 255
10670     LUTDATA0%(I)=(I*160)/255:LUTDATA1%(I)=(I*192)/255
10680   NEXT
10690   SETDLUT 0,LUTDATA0%:SETDLUT 1,LUTDATA1%
10700   ERASE LUTDATA0%,LUTDATA1%

```

```

10710   FOR I=0 TO 6
10720       SETDLVL I
10730   NEXT
10740   DISPLAY 16:WDISP -1,0,0:BACKDISP 0
10750 RETURN
10760 '-----
10770 'Multiple window coordinate, area discrimination value settings
10780 '-----
10790 *MWINDSET
10800   FILTERIN 1:CLS 3:BACKDISP 1,0 'Display window only
10810   WDISP -1,0,0:WDISP 0,1,1      'Display plane 0
10820   LEVEL -1,511,511
10830   RMODE 0,0,1:MEASURE:LABEL:LSORT 8
10840   W.MAX=LNUM
10850   IF W.MAX<1 THEN
10860       PRINT "WINDOWS ARE NOT SET."
10870       END
10880   END IF
10890   FOR I=1 TO W.MAX      'Obtain circumscribed coordinates
10900       W.WA%(I-1)=LDATA(I,0)
10910       W.X1%(I-1)=LDATA(I,7):W.Y1%(I-1)=LDATA(I,8)
10920       W.X2%(I-1)=LDATA(I,9):W.Y2%(I-1)=LDATA(I,10)
10930   NEXT
10940   MWSET 0, W.MAX, W.X1%, W.Y1%, W.X2%, W.Y2%
10950   FILTERIN 0:CLS 3:BACKDISP 0
10960   WDISP -1,1,1
10970   SCNLUT W.SCN      'Returns binary level
10980   FOR I=0 TO W.MAX-1 'Sets the area discrimination value
10990       W.LO%(I)=W.WA%(I)*W.LO/100
11090       W.HI%(I)=W.WA%(I)*W.HI/100+1
11010   NEXT
11020 RETURN
11030 '-----
11040 'STEP interrupt process
11050 '-----
11060 *INTR1
11070   IF PIN(5) THEN
11080       GOSUB *STANDMEAS      'Reference registration
11090   ELSE
11110       GOSUB *MEAS          'Measurement
11110   END IF
11120 RETURN
11130 '-----
11140 'Console Key interrupt process
11150 '-----
11160 *INTR2

```

```
11170     K=KEYIN(0)
11180         IF K=&H90 THEN GOSUB *STANDMEAS 'Shift+Enter Reference regist.
11190         IF K=&H10 THEN GOSUB *MEAS      'Enter Measurement
11200 RETURN
11210 '-----
11220 'Reference image registration (Records center of gravity coordinates for position compensation.)
11230 '-----
11240 *STANDMEAS
11250     WSCROLL 0,0
11260     MEASURE
11270     W.SX=MDATA(7,2)
11280     W.SY=MDATA(7,4)
11290     LOCATE 0,0:PRINT "Reference registration completed."
11300 RETURN
11310 '-----
11320 'Measurement
11330 '-----
11340 *MEAS
11350     TIM0%=TIMER
11360     WSCROLL 0,0           'Moves to window's origin.
11370     EVENTIN 1,0,1:MEASURE 'Center of gravity measurement
11380     W.DX=MDATA(7,2)-W.SX  'Position compensation calculation
11390     W.DY=MDATA(7,4)-W.SY
11400     WSCROLL W.DX, W.DY   'Position compensation
11410     MWMEAS 0, 0, W.BW0, 1, W.DX, W.DY, 0, W.AR%
11420     ARRAYFUNC 11, W.MAX, W.JG&, W.AR%, W.LO%, W.HI%
11430     TIM%=TIMER-TIM0&
11440     LOCATE 0,0:PRINT USING "Processing time=####ms";TIM%*10
11450     GOSUB *JUDGDISP
11460     GOSUB *JUDGOUT
11470 RETURN
11480 '-----
11490 'Discrimination results display process
11500 '-----
11510 *JUDGDISP
11520     FOR I=0 TO W.MAX-1
11530         J=W.JG7(FIX(I/32)) AND 2^(I MOD 32 )
11540         IF J THEN
11550             COLOR 0:J$="OK"
11560         ELSE
11570             COLOR 7:J$="NG"
11580         END IF
11590         PRINT USING "No ### @";I,J$
11600         COLOR 0
11610     NEXT
11620 RETURN
```

```

11630 '-----
11640 'Discrimination results terminal output process
11650 '-----
11660 *JUDGOUT
11670   FOR I=0 TO FIX((W.MAX-1)/32)
11680     DOUT W.JG&(I),0,32
11690   NEXT
11700 RETURN

```

## 3-2 Shape Inspection using Window Enlargement/Reduction

### 3-2-1 Introduction

The window enlargement/reduction functions are used to determine whether a work piece's shape matches the shape of a reference work piece. Perform the operations below in menu mode beforehand.

- 1, 2, 3...**
1. Set up a window in window plane 0 for use as a measurement region. (Do not specify the entire screen, however.)
  2. Set the binary level.

Once the program has been executed, the following operations can be used to determine if a work piece is acceptable (OK) or unacceptable (NG):

#### Measurement

Measure when the Console's Enter Key is pressed or the STEP signal goes ON. If the work piece is larger than the reference piece, the blurred area is measured. If the work piece is smaller than the reference piece, omission inspection is measured. The blurred area and omission area are compared to the reference area to determine whether the work piece is acceptable or not.

**Reference Image Registration** Register the reference image when the Console's Shift+Enter Keys are pressed or DI5 is ON and the STEP signal goes ON. A window for blurring inspection is automatically drawn in window plane 7, and a window for omission inspection is automatically drawn in window plane 6.

### 3-2-2 Program

```

10000 '*****
10010 '                (c)Copyright OMRON Co. 1992
10020 '                All Rights Reserved
10030 '-----
10040 '                F300 Sample Program
10050 ' Shape Inspection using Window Zooming
10060 ' Useful for operations such as character identification.
10070 ' Before program execution, set up (in menu mode) a window in
10080 ' window plane 0 for use as a processing region.
10090 '     Window plane 7: Blurring inspection
10100 '     Window plane 6: Omission inspection
10110 '     Window plane 0: Processing region
10120 '*****
10130 DCNT=3           'Number of enlargements
10140 ECNT=1           'Number of reductions
10150 JDG7=10         'Number of pixels for determining blurring

```

```

10160 JDG6=10          'Number of pixels for determining omission
10170 BW=0            'Pixels measured (0=black, 1=white)
10180 '-----
10190 'Main process
10200 '-----
10210 *MAIN
10220   GOSUB *HWINIT
10230   GOSUB *INITIAL
10240   GOSUB *WINDAREA
10250   GOSUB *WINDSET
10260   ON INTR GOSUB *INTR1:INTR ON
10270   ON CONKEY GOSUB *INTR2:CONKEY ON
10280   WHILE (1)
10290   WEND
10300 '-----
10310 'Initialization process
10320 '-----
10330 *INITIAL
10340   CONSOLE ,,0:LOCATE ,,0
10350   DISPLAY 31,0          'Binary display within window
10360   CAMMODE 1            'Output of frame buffer contents
10370   FLASH 2              'Shutter with STEP synchronization
10380   WINDOW 0            'Draws window of scene 0
10390   MMODE 7,BW,1        'Measurement mode for blurring
10400   MMODE 6,NOT BW,1    'Measurement mode for omission
10410 RETURN
10420 '-----
10430 'Establishment of the measurement processing region
10440 '-----
10450 *WINDAREA
10460   FILTERIN 1:BACKDISP 1,0          'Display window only
10470   WDISP -1,0,0:WDISP 0,1,1        'Display plane 0
10480   LEVEL -1,511,511
10490   RMODE 0,0,1:MEASURE:LABEL:LSORT 8
10500   W.MAX=LNUM
10510   IF W.MAX<1 THEN
10520     PRINT "WINDOWS ARE NOT SET."
10530     END
10540   END IF
10550   LSORT 0
10560   W.X1=LDATA(1,7):W.Y1=LDATA(1,8)  'Set measurement region.
10570   W.X2=LDATA(1,9):W.Y2=LDATA(1,10)
10580   BOX W.X1, W.Y1, W.X2, W.Y2, 1,,1
10590   FILTERIN 0:BACKDISP 0:SCNLUT 0
10600   WDISP -1,0,0:WDISP 6,1,1:WDISP 7,1,1
10610 RETURN

```

```

10620 '-----
10630 'Hardware initialization
10640 '-----
10650 *HWINIT
10660     CAMERA 0:CAMMODE 0:FLASH 1:FILTERIN 0:FILTER 0,1,0
10670     MASKBIT 2,0,0:MASKBIT 3,0,0:MASKBIT 4,0,0
10680     CLS 0:CLS 1:CLS 2:CLS 3:SBANK 1:CLS 4:SBANK 0:CLS 4
10690     LEVEL -1,128,255:WSCROLL 0,0:SSCROLL 0,0:WZOOM:SZOOM
10700     DIM LUTDATA0%(256),LUTDATA1%(256)
10710     FOR I=0 TO 255
10720         LUTDATA0%(I)=(I*160)/255:LUTDATA1%(I)=(I*192)/255:
10730     NEXT
10740     SETDLUT 0,LUTDATA0%:SETDLUT 1,LUTDATA1%
10750     ERASE LUTDATA0%,LUTDATA1%
10760     FOR I=0 TO 6
10770         SETDLVL I
10780     NEXT
10790     DISPLAY 16:WDISP -1,0,0:BACKDISP 0
10800 RETURN
10810 '-----
10820 'STEP interrupt process
10830 '-----
10840 *INTR1
10850     IF PIN(5) THEN
10860         GOSUB *WINDSET           'Window setup
10870     ELSE
10880         GOSUB *MEAS             'Measurement
10890     END IF
10900 RETURN
10910 '-----
10920 'Console Key interrupt process
10930 '-----
10940 *INTR2
10950     K=KEYIN(0)
10960     IF K=&H90 THEN GOSUB *WINDSET     'Shift+Enter Window setup
10970     IF K=&H10 THEN GOSUB *MEAS       'Enter Measurement
10980 RETURN
10990 '-----
11090 'Measurement window setup
11010 '-----
11020 *WINDSET
11030     MASKBIT 2,0,1:CLS 2:MASKBIT 2,0,0     'Deletes other than window 0
11040     *LOOP
11050     RMODE 0,BW,1
11060     MEASURE
11070     LABEL

```

```
11080   IF LNUM<1 THEN PRINT "ERRO":GOTO *LLOOP
11090   FOR I=1 TO LNUM
11100       LPUTIMG I,2,0,7           'Draws figure in window mem.
11110   NEXT
11120   WDILA 7,W.X1, W.Y1, W.X2, W.Y2,,DCNT
11130   MASKBIT 2,0,&H7F           'Masks other than plane 7
11140   BOX W.X1, W.Y1, W.X2, W.Y2,2,0,XOR 'Reverses window memory
11150   MASKBIT 2,0,0
11160   FOR I=1 TO LNUM
11170       LPUTIMG I,2,0,6           'Draws figure in window mem.
11180   NEXT
11190   WEROS 6,W.X1, W.Y1, W.X2, W.Y2,,ECNT
11200 RETURN
11210 '-----
11220 'Measurement process
11230 '-----
11240 *MEAS
11250   LOCATE 0,0
11260   MEASURE
11270   A7=MDATA(7,0)
11280   IF A7>JDG7 THEN J7$="NG" ELSE J7$="OK"
11290   A6=MDATA(6,0)
11300   IF A6>JDG6 THEN J6$="NG" ELSE J6$="OK"
11310   PRINT USING "Blurred pixels=##### @":A7,J7$
11320   PRINT USING "Omitted pixels=##### @":A6,J6$
11330 RETURN
```



## Appendix A

### Table of Error Messages

Error message		Error code
A	Access denied	85
	Application error (polish)	101
	Application error (accum)	102
B	Bad drive specification	70
	Bad file name	56
	Bad file number	52
C	Can't continue	17
	CASE without END SELECT	124
	CASE without SELECT	121
	Channel number error	202
	Checksum error	117
	Compile timing error	119
D	DEF FN without END DEF	134
	Direct statement in file	57
	Disk full	68
	Disk I/O error	64
	Disk offline	62
	Division by zero	11
	DO without LOOP	135
	Duplicate definition	10
	Duplicate label	31
E	ELSE IF without END IF	129
	ELSE IF without IF	125
	ELSE without END IF	130
	ELSE without IF	126
	END DEF without DEF FN	120
	END IF without IF	127
	END SELECT without SELECT	122
	END SUB without SUB	133
	EXIT without DEF	138
	EXIT without DO	141
	EXIT without FOR	140
	EXIT without SUB	139
F	Feature not available	33
	FIELD overflow	50
	File already exist	65
	File already open	54
	File not found	53
	File not open	60
	File write protect	61
	FOR without NEXT	26
	I	IF without END IF
Illegal direct		12
Illegal function call		5
Illegal jump		137
Input past end		55
L	Line buffer overflow	23
	Local variable overflow	131
	LOOP without DO	136
M	Missing operand	22
N	NEXT without FOR	1
	No RESUME	19

Error message		Error code
O	Out of compile work space	110
	Out of DATA	4
	Out of memory	7
	Out of string space	14
	OV	100
	Overflow	6
P	Parameter overflow	118
	Path not found	76
	Path/File access error	75
R	Rename across disks	73
	RESUME without error	20
	RETURN without GOSUB	3
S	SELECT without END SELECT	123
	Sequential I/O only	59
	String too long	15
	SUB without END SUB	132
	Subscript out of range	9
	Syntax error	2
T	Type mismatch	13
U	Undefined label	32
	Undefined line number	8
	Undefined user function	18
	Unit error	201
	Unprintable error	99
V	Vision error	200
W	WEND without WHILE	30
	WHILE without WEND	29

## Appendix B Reserved Words

Reserved words				
A	ABS ARC ATTR\$	AKCNV\$ AS AUTO	ALL ASC AUTOLVL	APPEND ATN
B	BACKDISP BEEP BUSY	BASE BLOAD	BCOPY BOX	BCOPY2 BSAVE
C	CALL CANCEL CHANGE CIRCLE CLS COMMON COS CVD	CAMCHK CASE CHDIR CLEAR COLOR CONSOLE CSNG CVI	CAMERA CDBL CHR\$ CLNG COLOR@ CONT CSRLIN CVL	CAMMODE CHAIN CINT CLOSE COM COPY CURSOR CVS
D	DATA DEFDBL DEFSTR DGOTO DNEXT DSKF	DATE\$ DEFINE DELETE DIM DO	DCASE DEFLNG DENY DIN DOUT	DEF DEFSNG DEVICE DISPLAY DSA
E	EDIT END ERL ERROUT	ELLIPSE ENHANCE ERR EVENTIN	ELSE EOF ERRMSG EXIT	ELSEIF ERASE ERROR EXP
F	FIELD FILTERIN FN	FILES FIND FOR	FILTDATA FIX FRE	FILTER FLASH
G	GATE GET@ GETLUT	GCOPY GETBLUT GOSUB	GCOPY2 GETDLUT GO	GET GETDLVL GOTO
H	HELP	HEX\$	HISTGRAM	
I	IF INPUT INTR	IMGLOAD INPUT\$ IPL	IMGSAVE INSTR	INKEY\$ INT
J	JIS\$			
K	KACNV\$ KILL KMID\$ KTYPE	KEXT\$ KINPUT KNJ\$	KEY KINSTR KPLOAD	KEYIN KLEN KPOS
L	LABLE LEFT\$ LFILES LLIST LOCATE LPOINT LSET	LBOUND LEN LHELP LNUM LOF LPOS LSORT	LCASE\$ LET LINE LOAD LOG LPRINT LTRIM\$	LDATA LEVEL LIST LOC LOOP LPUTIMG
M	MASKBIT MENU MKD\$ MMODE	MDATA MERGE MKI\$ MON	MDATA2 MID\$ MKL\$	MEASURE MKDIR MKS\$
N	NAME	NEW	NPIECE	

Reserved words				
O	OCT\$ OPTION	ON OUTPUT	OFF	OPEN
P	PEEK POKE POUT PUT@	PIECE\$ POLYGON PRINT	PIN POLYLINE PSET	POINT POS PUT
R	RANDOMIZE RENUM RESUME RMODE RTRIM\$	RDATA REPEAT RETURN RND RUN	READ REPLACE RIGHT\$ RNEXT RUNL	REM RESTORE RMDIR RSET
S	SAVE SCNCAM SCNLOAD SDATA2 SET SETLUT SIN SPLINE STOP STRMODE	SBACK SCNCALIB SCNLUT SEARCH SETBLUT SFTBLUT SPACE\$ SQR STR\$ SUB	SCAN SCNJUDG SCNSAVE SEGPTR SETDLUT SFTLUT SPC SSCROLL STRCHK SWAP	SCANSET SCNLEVEL SDATA1 SELECT SETDLVL SGN SPCLOSE STEP STRING\$ SYSTEM
T	TAB THEN	TAN TO	TIME\$ TROFF	TIMER TRON
U	UBOUND USR USR3 USR7	UCASE\$ USR0 USR4 USR8	UNTIL USR1 USR5 USR9	USING USR2 USR6
V	VAL VENUS	VARPTR	VDWAIT	VIDEOIN
W	WAIT WIDTH WSCROLL	WDISP WINDOW	WEND WPBIT	WHILE WRITE

# Appendix C

## Induction Functions

(P# =  $\pi$  = 3.14159265358979)

Function	Induction formula
log a X	LOG(X)/LOG(a)
sec X	1/COS(X)
cosec X	1/SIN(X)
cot X	1/TAN(X)
$\sin^{-1} X$	ATN(X/SQR(-X*X+1))
$\cos^{-1} X$	-ATN(X/SQR(-X*X+1))+P#/2
$\sec^{-1} X$	ATN(SQR(X*X-1)+(SGN(X)-1)*P#/2
$\csc^{-1} X$	ATN(1/SQR(X*X-1)+(SGN(X)-1)*P#/2
$\cot^{-1} X$	-ATN(X)+P#/2
sinh X	(EXP(X)-EXP(-X))/2
cosh X	(EXP(X)+EXP(-X))/2
tanh X	-EXP(-X)/(EXP(X)+EXP(-X))*2+1
sech X	2/(EXP(X)+EXP(-X))
cosech X	2/(EXP(X)-EXP(-X))
coth X	EXP(-X)/(EXP(X)-EXP(-X))*2+1
$\sinh^{-1} X$	LOG(X+SQR(X*X+1))
$\cosh^{-1} X$	LOG(X+SQR(X*X-1))
$\tanh^{-1} X$	LOG((1+X)/(1-X))/2
$\operatorname{sech}^{-1} X$	LOG(SQR(-X*X+1)+1)+1/X
$\operatorname{cosech}^{-1} X$	LOG((SGN(X)*SQR(X*X+1)+1)/X)
$\operatorname{coth}^{-1} X$	LOG((X+1)/(X-1))/2

# Index

## A

ABS, 54  
AKCNV\$, 54  
alphabetical listing, of version 2.00 functions & commands, 171  
ARC, 54  
arguments, 31  
array operations, using ARRAYFUNC, 175, 184  
ARRAYFUNC, 184  
ASC, 55  
ATN, 55  
ATTR, 55  
AUTO, 56  
AUTOLVL, 56

## B

BACKDISP, 56  
BATCHK, 185  
BCD, converting to binary, 180, 185  
BCDTOBIN, 185  
BCOPY, 57  
BCOPY2, 58  
BEDGE, 185  
BEEP, 58  
binary, converting to BCD, 180, 186  
binary image planes, memory, 43  
binary images  
  deleting exterior graphics, 188  
  eliminating isolated points, 188  
  enlarging in window memory, 200  
  expanding, 187  
  filling holes in, 193  
  logic operations between, 175, 186  
  processing, 174  
  reducing, 189  
  reducing in window memory, 201  
  reducing line thickness, 200  
  setting binary level, 195  
binary level  
  setting, 195  
  setting range with version 2.00, 182  
BINTOBCD, 186  
block diagrams, 40  
BMFUNC, 186  
boot-up mode, setting, 194

BOX, 59  
BUSY, 59

## C

CALL, 60  
CAMCHK, 60  
CAMERA, 60  
camera synchronization, setting with CAMSYNC, 181, 186  
CAMMODE, 60  
CAMSYNC, 186  
CDBL, 61  
CHAIN, 61  
CHANGE, 62  
character strings, 21  
characters, 20  
CHDIR, 62  
CHR\$, 63  
CINT, 63  
CIRCLE, 64  
CLEAR, 65  
CLNG, 65  
CLOSE, 65  
CLS, 66  
COLOR, 66  
COLOR@, 66  
COM ON/OFF/STOP, 67  
command  
  ARC, 54  
  ARRAYFUNC, 184  
  AUTO, 56  
  BACKDISP, 56  
  BCOPY, 57  
  BCOPY2, 58  
  BEDGE, 185  
  BEEP, 58  
  BMFUNC, 186  
  BOX, 59  
  BUSY, 59  
  CALL, 60  
  CAMERA, 60  
  CAMMODE, 60  
  CAMSYNC, 186  
  CHAIN, 61  
  CHANGE, 62  
  CHDIR, 62  
  CIRCLE, 64  
  CLEAR, 65  
  CLOSE, 65

CLS, 66  
COLOR, 66  
COLOR@, 66  
COM ON/OFF/STOP, 67  
COMMON, 67  
CONKEY ON/OFF/STOP, 187  
CONSOLE, 68  
CONT, 68  
CURSOR, 69  
DATA, 72  
DEF FN, 73  
DEF FN...END DEF, 74  
DEFDBL, 74  
DEFINT, 74  
DEFLNG, 75  
DEFSNG, 75  
DEFSTR, 75  
DELETE, 76  
DEVICE, 76  
DILA, 187  
DIM, 76  
DISPLAY, 77  
DO REPEAT-LOOP, 79  
DO UNTIL-LOOP, 80  
DO WHILE-LOOP, 80  
DO-LOOP REPEAT, 78  
DO-LOOP UNTIL, 78  
DO-LOOP WHILE, 79  
DOUT, 80  
EDGRJECT, 188  
EDIT, 81  
ELIM, 188  
ELLIPSE, 82  
END, 82  
ENHANCE, 83  
ERASE, 83  
EROS, 189  
ERRMSG, 84  
ERROR, 84  
ERROUT, 85  
EVENTIN, 85  
EXIT DEF/DO/FOR/SUB, 86  
FIELD #, 87  
FILES, 88  
FILTDATA, 88, 189  
FILTER, 88  
FILTERIN, 89  
FILTSEL, 190  
FIND, 89  
FLASH, 90  
FOR..TO..STEP-NEXT, 91  
FORMAT, 191  
GATE, 91  
GCOPY, 92  
GCOPY2, 92  
GET #, 93  
GET@, 93  
GETBLUT, 94  
GETDLUT, 94  
GETDLVL, 95  
GETLUT, 95  
GETMDATA, 191  
GMFUNC, 193  
GOSUB, 95  
GOTO, 96  
HELP, 96  
HELP ON/OFF/STOP, 96  
HFILL, 193  
HISTGRAM, 97  
IF..GOTO-ELSE, 98  
IF..THEN-ELSE, 98  
IF..THEN-ELSEIF-ELSE-END IF, 99  
IMGLOAD, 99  
IMGSAVE, 100  
INPUT, 101  
INPUT WAIT, 102  
INPUT#, 101  
INTR ON/OFF/STOP, 104  
IPL, 104, 194  
KEY, 106  
KEY LIST, 106  
KEY ON/OFF/STOP, 106  
KILL, 107  
KINPUT, 108  
KPLOAD, 110  
LABEL, 111  
LET, 113  
LEVEL, 114, 195  
LINE, 114  
LINE INPUT, 115  
LINE INPUT WAIT, 116  
LINE INPUT#, 116  
LIST, 116  
LOAD, 117  
LOCATE, 118  
LPUTIMG, 119  
LSET, 120  
LSORT, 120, 195  
MASK3, 195  
MASKBIT, 120  
MEASURE, 122  
MENU, 122  
MERGE, 122  
MKDIR, 124  
MMODE, 125  
MWMEAS, 196  
MWSET, 197  
NAME, 126  
NEW, 126  
ON COM GOSUB, 127  
ON CONKEY GOSUB, 197  
ON ERROR GOTO, 128  
ON GOSUB, 130  
ON GOTO, 131  
ON HELP GOSUB, 128  
ON INTR GOSUB, 128  
ON KEY GOSUB, 129  
ON STOP GOSUB, 129  
ON TIMES\$ GOSUB, 130  
OPEN(1), 131  
OPEN(2), 132  
OPTION BASE, 132  
POLYGON, 134  
POLYLINE, 134  
POUT, 135  
PRINT, 135  
PRINT #, 137  
PRINT USING, 136  
PRINT# USING, 138

- PSET, 138
- PUT #, 139
- PUT@, 139
- RANDOMIZE, 140
- READ, 140
- REM, 141
- RENUM, 141
- REPLACE, 141
- RESTORE, 142
- RESUME, 142
- RETURN, 142
- RMDIR, 143
- RMODE, 144
- RSET, 145
- RUN, 145
- SAVE, 146
- SBANK, 146
- SCAN, 147
- SCANSET, 147
- SCNCALIB, 148
- SCNLOAD, 149
- SCNLUT, 149
- SCNSAVE, 149
- SELECT...CASE-CASEELSE-END SELECT, 151
- SET, 151
- SETBLUT, 152
- SETDLUT, 152
- SETDLVL, 153
- SETLUT, 153
- SFTBLUT, 153
- SFTLUT, 154
- SOBEL, 199
- SPCLOSE, 157
- SPLINE, 157
- SSCROLL, 158
- STOP, 158
- STOP ON/OFF/STOP, 159
- STRMODE, 160
- SUB-END SUB, 161
- SWAP, 161
- SZOOM, 200
- THIN, 200
- TIMES ON/OFF/STOP, 163
- TROFF, 164
- TRON, 164
- VDWAIT, 165
- VIDEOIN, 165
- WDILA, 200
- WDISP, 166
- WEROS, 201
- WHILE-WEND, 166
- WINDOW, 167
- WRITE, 167
- WRITE #, 168
- WSCROLL, 168
- WZOOM, 201
  
- command-function, DATE\$, 73
- COMMON, 67
- compile work space, changes in version 2.00, 172
- CONKEY ON/OFF/STOP, 187
- CONSOLE, 68
  
- console key, interrupts, 180, 187, 197
- constants, 21
  - character, 21
  - character strings, 21
  - numeric, 22
    - integer, 22
    - long integer, 22
    - real number, 23
- CONT, 68
- conversion, numeric data, 25
- COS, 69
- criteria, checking with SCNJUDGE, 177, 198
- CSNG, 69
- CSRLIN, 69
- CURSOR, 69
- CVD, 70
- CVI, 71
- CVL, 71
- CVS, 71
  
- D**
- DATA, 72
- data, measured, flow, 41
- DATE\$, 73
- DEF FN, 73
- DEF FN...END DEF, 74
- DEFDBL, 74
- DEFINT, 74
- DEFLNG, 75
- DEFSNG, 75
- DEFSTR, 75
- DELETE, 76
- DEVICE, 76
- DILA, 187
- DIM, 76
- DIN, 77
- DISPLAY, 77
- display LUT, 39
- DO REPEAT-LOOP, 79
- DO UNTIL-LOOP, 80
- DO WHILE-LOOP, 80
- DO-LOOP REPEAT, 78
- DO-LOOP UNTIL, 78
- DO-LOOP WHILE, 79
- DOUT, 80
- drawing density, 46



drawing mode, 46  
DSA, 81  
DSKF, 81

## **E**

EDGRJECT, 188  
EDIT, 81  
ELIM, 188  
ELLIPSE, 82  
END, 82  
ENHANCE, 83  
EOF, 83  
ERASE, 83  
ERL, 84  
EROS, 189  
ERR, 84  
ERRMSG, 84  
ERROR, 84  
error messages, table, 213  
ERROUT, 85  
EVENTIN, 85  
EXIT DEF/DO/FOR/SUB, 86  
EXP, 87  
expressions, 29  
  character, 29  
  functions, 31  
  logical, 29  
  numeric, 29  
  relational, 29

## **F**

FIELD #, 87  
file delimiters, changes in version 2.00, 172  
FILES, 88  
fill measurement, 50  
FILTDATA, 88, 189  
FILTER, 88  
filter, 39  
  3x3 mask filtering, 195  
  selecting filtering functions, 177, 190  
  setting line filter factors, 189  
FILTERIN, 89  
FILTSEL, 190  
FIND, 89  
FIX, 90  
FLASH, 90

FOR..TO..STEP-NEXT, 91  
FORMAT, 191  
FRE, 91  
function  
  ABS, 54  
  AKCNV\$, 54  
  ASC, 55  
  ATN, 55  
  ATTR, 55  
  AUTOLVL, 56  
  BATCHK, 185  
  BCDTOBIN, 185  
  BINTOBCD, 186  
  CAMCHK, 60  
  CDBL, 61  
  CHR\$, 63  
  CINT, 63  
  CLNG, 65  
  COS, 69  
  CSNG, 69  
  CSRLIN, 69  
  CVD, 70  
  CVI, 71  
  CVL, 71  
  CVS, 71  
  DIN, 77  
  DSA, 81  
  DSKF, 81  
  EOF, 83  
  ERL, 84  
  ERR, 84  
  EXP, 87  
  FIX, 90  
  FRE, 91  
  GETVER, 192  
  HEX\$, 97  
  INKEY\$, 100  
  INPUT\$, 102  
  INSTR, 103  
  INT, 103  
  JIS\$, 105  
  KACNV\$, 105  
  KEXT\$, 105  
  KEYIN, 107  
  KINSTR, 108  
  KLEN, 108  
  KMID\$, 109  
  KNJS, 109  
  KPOS, 110  
  KTYPE, 111  
  LBOUND, 112  
  LCASE\$, 112  
  LDATA, 112  
  LEFT\$, 113  
  LEN, 113  
  LNUM, 117  
  LOC, 117  
  LOF, 118  
  LOG, 118  
  LPOINT, 119  
  LTRIM\$, 120  
  MDATA, 121  
  MDATA2, 121  
  MKD\$, 123

MKIS\$, 124  
MKL\$, 124  
MKSS\$, 125  
NPIECE, 126  
OCT\$, 127  
PIECE\$, 133  
PIN, 133  
POINT, 133  
POS, 135  
RDATA, 140  
RIGHT\$, 143  
RND, 144  
RTRIM\$, 145  
RUNL, 146  
RUNL2, 198  
SCNCAM, 148  
SCNJUDGE, 198  
SCNLEVEL, 148  
SCNSTAND, 199  
SDATA1, 149  
SDATA2, 150  
SEARCH, 150  
SGN, 155  
SIN, 155  
SPACE\$, 156  
SPC, 156  
SQR, 158  
STR\$, 159  
STRCHK, 160  
STRING\$, 160  
TAB, 161  
TAN, 162  
UBOUND, 164  
UCASE\$, 164  
VAL, 165

function, command  
MID\$, 123  
TIMES\$, 162  
TIMER, 163

functions, 31  
induction, 217

## G

GATE, 91  
GCOPY, 92  
GCOPY2, 92  
GET #, 93  
GET@, 93  
GETBLUT, 94  
GETDLUT, 94  
GETDLVL, 95  
GETLUT, 95  
GETMDATA, 191  
GETVER, 192  
GMFUNC, 193  
GOSUB, 95

GOTO, 96  
gray-scale images, logic operations between, 193

## H

HELP, 96  
HELP ON/OFF/STOP, 96  
HEX\$, 97  
HFILL, 193  
HISTGRAM, 97

## I

IF..GOTO-ELSE, 98  
IF..THEN-ELSE, 98  
IF..THEN-ELSEIF-ELSE END IF, 99  
image cut-off, 50  
images  
cameras  
display status, 40  
measurement status, 40  
data flow, 36  
display LUT, 39  
filter, 39  
LUT, 38  
memory  
display status, 41  
measurement status, 41  
VRAM, 36  
character memory, 36  
graphic memory, 37  
image memory, 37  
shading memory, 38  
window memory, 37

IMGLOAD, 99  
IMGSAVE, 100  
induction functions, 217  
INKEY\$, 100  
INPUT, 101  
INPUT WAIT, 102  
INPUT#, 101  
INPUT\$, 102  
inspection, for character blurring/omission, 174  
INSTR, 103  
INT, 103  
integer constants, 22  
format  
decimal, 22  
hexadecimal, 22  
octal, 22  
interrupts, 32  
console key, 180, 187, 197

INTR ON/OFF/STOP, 104

IPL, 104, 194

## J

JIS\$, 105

## K

KACNV\$, 105

KEXT\$, 105

KEY, 106

KEY LIST, 106

KEY ON/OFF/STOP, 106

KEYIN, 107

KILL, 107

KINPUT, 108

KINSTR, 108

KLEN, 108

KMID\$, 109

KNJ\$, 109

KPLOAD, 110

KPOS, 110

KTYPE, 111

## L

LABEL, 111

labelling, 48  
  renumbering labels, 176, 195

labels, 32

LBOUND, 112

LCASE\$, 112

LDATA, 112

LEFT\$, 113

LEN, 113

LET, 113

LEVEL, 114, 195

LINE, 114

LINE INPUT, 115

LINE INPUT WAIT, 116

LINE INPUT#, 116

lines

  definition, 20

  format, 20

  numbers, 20

  statements, 20

LIST, 116

LNUM, 117

LOAD, 117

LOC, 117

LOCATE, 118

LOF, 118

LOG, 118

long integer constants, 22  
  format

    decimal, 22

    hexadecimal, 23

    octal, 22

LPOINT, 119

LPUTIMG, 119

LSET, 120

LSORT, 120, 195

LTRIM\$, 120

LUT, 38

## M

mask bits, memory, 43

MASK3, 195

MASKBIT, 120

MDATA, 121

MDATA2, 121

MEASURE, 122

measurement

  fill, 50

  scan, 51

memory

  binary image planes, 43

  frame, 43

    image, 37

    shading, 38

    window, 37

  mask bits, 43

  plane, 43

    character, 36

    graphic, 37

Memory Card

  checking battery voltage, 180, 185

  formatting, 180, 191

MENU, 122

MERGE, 122

MID\$, 123

MKD\$, 123

MKDIR, 124

MKI\$, 124

MKL\$, 124

MKSS\$, 125

MMODE, 125  
multiple-window functions  
  measurements, 196  
  registering windows, 197  
  use in sample program, 204  
  version 2.00 addition, 172

MWMEAS, 196

MWSET, 197

## N

NAME, 126

NEW, 126

NPIECE, 126

## O

OCT\$, 127

ON COM GOSUB, 127

ON CONKEY GOSUB, 197

ON ERROR GOTO, 128

ON GOSUB, 130

ON GOTO, 131

ON HELP GOSUB, 128

ON INTR GOSUB, 128

ON KEY GOSUB, 129

ON STOP GOSUB, 129

ON TIMES GOSUB, 130

OPEN(1), 131

OPEN(2), 132

operations  
  priority, 32  
  screen, 35

operators, 27  
  arithmetic, 27  
  logical, 28  
  relational, 28

OPTION BASE, 132

## P

PIECES\$, 133

PIN, 133

planes  
  frame, 43  
  memory, 43

POINT, 133

POLYGON, 134

POLYLINE, 134

POS, 135

POUT, 135

PRINT, 135

PRINT #, 137

PRINT USING, 136

PRINT# USING, 138

programs, samples for version 2.00, 203

PSET, 138

PUT #, 139

PUT@, 139

## R

RANDOMIZE, 140

raw images  
  logic operations between, 175  
  processing, 175

RDATA, 140

READ, 140

real number constants, 23  
  format  
    double-precision, 23  
    single-precision, 23

reference image data, checking with SCNSTAND, 178, 199

REM, 141

RENUM, 141

REPLACE, 141

reserved words, table, 215

RESTORE, 142

RESUME, 142

RETURN, 142

RIGHT\$, 143

RMDIR, 143

RMODE, 144

RND, 144

RS-232C, standard I/O settings, 181

RSET, 145

RTRIM\$, 145

RUN, 145

run length, 47  
  detailed, 48  
  simple, 47  
  using for various calculations, 179, 198

RUNL, 146

RUNL2, 198

**S**

sample programs, for version 2.00, 203  
SAVE, 146  
SBANK, 146  
SCAN, 147  
scan measurement, 51  
SCANSET, 147  
SCNCALIB, 148  
SCNCAM, 148  
SCNJUDGE, 198  
SCNLEVEL, 148  
SCNLOAD, 149  
SCNLUT, 149  
SCNSAVE, 149  
SCNSTAND, 199  
scrolling, 49  
    shading, 49  
    window, 49  
SDATA1, 149  
SDATA2, 150  
SEARCH, 150  
SELECT...CASE-CASEELSE-END SELECT, 151  
SET, 151  
SETBLUT, 152  
SETDLUT, 152  
SETDLVL, 153  
SETLUT, 153  
SFTBLUT, 153  
SFTLUT, 154  
SGN, 155  
shading compensation  
    frame memory, 38  
    scrolling, 49  
shading memory, zooming, 200  
SIN, 155  
SOBEL, 199  
SPACE\$, 156  
SPC, 156  
SPCLOSE, 157  
specifications, changes in version 2.00, 172  
SPLINE, 157  
SQR, 158  
SSCROLL, 158  
statements, 20  
STOP, 158

STOP ON/OFF/STOP, 159  
STR\$, 159  
STRCHK, 160  
STRING\$, 160  
STRMODE, 160  
SUB-END SUB, 161  
SWAP, 161  
symbols, 20  
syntax, 19  
SZOOM, 200

**T**

TAB, 161  
TAN, 162  
THIN, 200  
TIMES\$, 162  
TIMES ON/OFF/STOP, 163  
TIMER, 163  
TROFF, 164  
TRON, 164

**U**

UBOUND, 164  
UCASE\$, 164

**V**

VAL, 165  
variables, 23  
    arrays, 24  
    names, 24  
    reserved words, 24  
    system, 31  
    type declarators, 23  
VDWAIT, 165  
version, obtaining with GETVER, 192  
version 2.00  
    alphabetical listing of functions & commands, 171  
    BCD/binary conversion, 180  
    binary image processing, 174  
    binary level setting range, 182  
    calculations using simple run length, 179  
    camera synchronization, 181  
    changes in specifications, 172  
    checking menu settings, 177  
    console key interrupts, 180  
    filtering selection, 177  
    high-speed array operations, 175  
    image-to-image calculations, 175

---

## *Index*

---

- loading/saving programs through RS-232C, 181
- memory card operations, 180
- multiple-window functions, 172
- obtaining system information, 181
- raw image processing, 175
- renumbering labels, 176
- standard I/O settings, 181
- summary of additional capabilities, 170
- window enlargement/reduction, 174
- zooming window & shading memory, 176

VIDEOIN, 165

VRAM, 36

- character memory, 36
- graphic memory, 37
- image memory, 37
- shading memory, 38
- window memory, 37

## **W**

WDILA, 200

WDISP, 166

WEROS, 201

WHILE-WEND, 166

WINDOW, 167

window memory, zooming, 201

windows, 44

- frame memory, 37
- paint, 45
- pattern matching, 45
- planes, 44
- scrolling, 49
- zooming, 176

WRITE, 167


WRITE #, 168

WSCROLL, 168

WZOOM, 201

## Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. Z92-E1-2  
 Revision code

The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	October 1992	Original production
1A	March 1993	<p><b>Page 38:</b> VRAM3 description has been changed from window memory to image memory.</p> <p><b>Pages 67, 68:</b> Sentence added to COMMON example section and the program example has been altered.</p> <p><b>Page 76:</b> Format of DIM has been corrected.</p> <p><b>Page 77:</b> Last sentence of DIM description has been corrected.</p> <p><b>Page 100:</b> Compression function specification has been corrected for IMGSAVE.</p> <p><b>Pages 121, 122:</b> Description for MDATA2 has been changed.</p> <p><b>Page 132:</b> S parameter has been added to the table.</p> <p><b>Page 173:</b> Induction formulas have been changed for functions cosec<sup>-1</sup> X, coth X, tanh<sup>-1</sup> X, sech<sup>-1</sup> X, and coth<sup>-1</sup> X.</p>
2	March 1994	<p>Version 2.00 of the F300 Visual Inspection System (OVL) added as Part II, Sections 1 to 3.</p> <p><b>Page 73:</b> Format for the description corrected.</p> <p><b>Page 74:</b> DEF FN . . . END DEF example corrected.</p> <p><b>Page 87:</b> Middle paragraph for EXP description corrected.</p> <p><b>Page 102:</b> Middle paragraph for INPUT WAIT description corrected.</p> <p><b>Page 116:</b> Middle paragraph for LINE INPUT WAIT description corrected. LINE INPUT# description corrected.</p> <p><b>Page 142:</b> Middle paragraph for RESUME description corrected.</p> <p><b>Page 149:</b> Format and middle paragraph description corrected for SCNLOAD .</p> <p><b>Page 151:</b> Header and format for SELECT . . . CASE-CASE ELSE-END SELECT corrected.</p> <p><b>Page 160:</b> Format for STRING\$ corrected.</p>