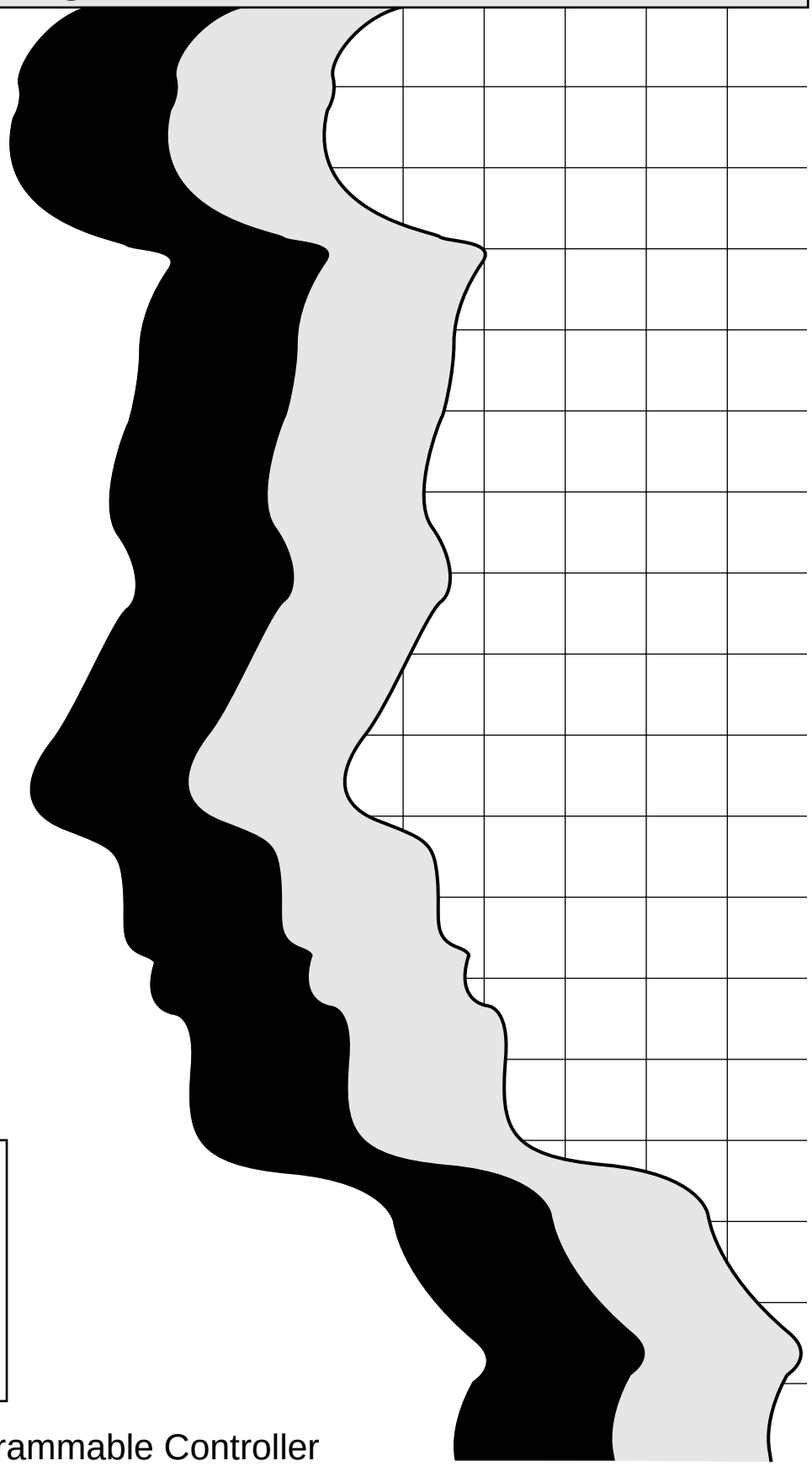
MITSUBISHI

type AD51H-BASIC

Programming Manual



A

MELSEC

Mitsubishi Programmable Controller

REVISIONS

*The manual number is given on the bottom left of the back cover.

Print Date	*Manual Number	Revision
Jan., 1992	IB (NA) 66323-A	First edition
Feb., 1993	IB (NA) 66323-B	Addition REFERENCE MANUAL, Section 11 (Command and Function : ZCNTL (code 80)), 12, APP.4.4.3, APP.5 Correction CONTENTS, Section 1, 2.1, 2.3, 2.8.3, 2.10.3, 3, 3.1, 3.3.1, 3.3.2, 4.2.6, 5.1, 7.2.1, 7.3.1, 7.3.4, 8.1, 8.2, 10.2, 11 (Command and Function : ATN, BSWAP, CHAIN, CLEAR, CONSOLE, CONT, CSNI, DATA, DATE$, DEFFN, DEFSTR, DEFZEVENT, ERASE, FORMAT, FRE, GETMEM, INKEY$, INPUT$, INSTR, INT, KEYLIST, KILL, LINE INPUT#, LOAD, LOCATE, LOF, MERGE, NAME, ON COM GOSUB, ON ERROR GOTO, PCRD (code 7, 12), PCWT (code 7, 8, 12), PRINT, PUTMEM, RDSET, RENUM, RIGHT$, RND, RUN (version 1), SAVE, SEARCH, SHT, STOP, SYSTEM, TAB, WTSET, ZCNTL, ZMESSAGE, ZMESSAGE GET, ZMESSAGE PUT, ZMOVE, ZRELEASE, ZRESERVE, ZSIGNAL, ZSTART), APP.1.1, APP.3, APP.4.4.1, APP.4.4.2, APP.6, APP.7.2.5

INTRODUCTION

Thank you for choosing the Mitsubishi MELSEC-A Series of General Purpose Programmable Controllers. Please read this manual carefully so that the equipment is used to its optimum. A copy of this manual should be forwarded to the end User.

CONTENTS

1. GENERAL DESCRIPTION ..... 1 - 1 ~ 1 - 2

1.1 Features ..... 1 - 1
1.2 Symbols Used in This Manual ..... 1 - 2

2. INTRODUCTION TO AD51H - BASIC ..... 2 - 1 ~ 2 - 27

2.1 Preparations for Using AD51H - BASIC ..... 2 - 1
2.2 Direct Mode and Program Mode ..... 2 - 4
2.3 Line Format ..... 2 - 4
2.4 Space and Key Word ..... 2 - 5
2.5 Characters that Can be Used in a BASIC Program ..... 2 - 7
2.6 Commands and Functions ..... 2 - 10
2.7 Constants ..... 2 - 11
2.7.1 Character string constants ..... 2 - 11
2.7.2 Numerical constants ..... 2 - 11
2.7.3 Single - precision and double - precision constants ..... 2 - 12
2.8 Variables ..... 2 - 14
2.8.1 Variable names and type - declaration characters ..... 2 - 14
2.8.2 Array variables ..... 2 - 15
2.8.3 Special variables (how to use B@ and W@) ..... 2 - 16
2.9 Type Conversion ..... 2 - 19
2.10 Expressions and Operators ..... 2 - 21
2.10.1 Arithmetic operators ..... 2 - 21
2.10.2 Comparison operators ..... 2 - 23
2.10.3 Logical operator ..... 2 - 24
2.11 Character String Operations ..... 2 - 26
2.12 Operation Precedence ..... 2 - 27

3. HOW TO CREATE AND EXCUTE A PROGRAM ..... 3 - 1 ~ 3 - 50

3.1 Program Input ..... 3 - 3
3.2 Executing and Correcting a Program ..... 3 - 5
3.2.1 Executing a program ..... 3 - 5
3.2.2 If an error occurs ..... 3 - 5
3.2.3 Correcting a program ..... 3 - 6
3.3 String and Reading a Program ..... 3 - 10
3.3.1 Memory cards that can be used for the AD51H-BASIC ..... 3 - 10
3.3.2 String a program ..... 3 - 11
3.3.3 Reading a program ..... 3 - 14
3.4 Arranging Files in the Memory Card and FD (File Order) ..... 3 - 15
3.4.1 Displaying file names ..... 3 - 15
3.4.2 Changing a file name ..... 3 - 17
3.4.3 Deleting a file ..... 3 - 18

3.5 Data Setting . . . . . 3 - 19
3.5.1 Assignment sentence . . . . . 3 - 19
3.5.2 Generating data in batch . . . . . 3 - 20
3.6 Jump and Repeat . . . . . 3 - 22
3.6.1 Unconditional jump . . . . . 3 - 22
3.6.2 Conditional jump . . . . . 3 - 23
3.6.3 Repeating a series of commands for a specified number of times . . . . . 3 - 24
3.6.4 Repeating a series of commands while a condition is satisfied . . . . . 3 - 26
3.7 Using a BASIC Conditional Sentence . . . . . 3 - 27
3.7.1 Specifying conditions . . . . . 3 - 27
3.7.2 Branch Command . . . . . 3 - 29
3.8 Using an Array . . . . . 3 - 31
3.8.1 Number of array dimensions . . . . . 3 - 32
3.9 Using a Subroutine . . . . . 3 - 33
3.10 Displaying a Character on the Screen . . . . . 3 - 34
3.10.1 Functions to display a character . . . . . 3 - 35
3.10.2 Displaying a character at a given position . . . . . 3 - 38
3.11 Data Input from the Keyboard . . . . . 3 - 40
3.12 Printout (Using a Printer) . . . . . 3 - 43
3.13 Character that are Processed in a BASIC Program . . . . . 3 - 44
3.13.1 Character type . . . . . 3 - 44
3.13.2 Processing in units of characters . . . . . 3 - 44
3.14 Numerical Functions . . . . . 3 - 48
3.15 Dividing and Executing Very Large Programs . . . . . 3 - 49

4. DATA TRANSFER WITH THE SEQUENCER AND WITH BUFFER MEMORY . . . . . 4 - 1 ~ 4 - 16

4.1 PC CPU Numerical Data and BASIC Numerical Data . . . . . 4 - 1
4.2 Data Communications between a PC CPU and BASIC . . . . . 4 - 2
4.2.1 Control table . . . . . 4 - 2
4.2.2 PC CPU station numbers . . . . . 4 - 3
4.2.3 Selecting the processing mode . . . . . 4 - 5
4.2.4 Bit/Word specification . . . . . 4 - 5
4.2.5 Specifying a device number . . . . . 4 - 6
4.2.6 Read/Write data storage areas . . . . . 4 - 8
4.3 Data Communications with the Buffer . . . . . 4 - 15

5. TRANSFERRING WITH GENERAL USE I/O . . . . . 5 - 1 ~ 5 - 7

5.1 AD51H to PC CPU (Input Device X) . . . . . 5 - 1
5.2 PC CPU to AD51H (Output Device Y) . . . . . 5 - 4

6. DATA FILE I/O PROCEDURES . . . . . 6 - 1 ~ 6 - 13

6.1 File Numbers . . . . . 6 - 2
6.2 Sequential File I/O Procedures . . . . . 6 - 3
6.3 Random File I/O Procedures . . . . . 6 - 7

6.4 Data File Handling Dangers ..... 6 - 12
6.4.1 Data file handling under multitasking ..... 6 - 12
6.4.2 The number of files that can be handled by each program ..... 6 - 13

7. COMMUNICATING WITH EXTERNAL DEVICES ..... 7 - 1 ~ 7 - 14

7.1 Correspondence of Interface to Channel Number ..... 7 - 1
7.2 Preparations for Communicating Data ..... 7 - 2
7.2.1 Setting communications parameters ..... 7 - 2
7.2.2 Control table ..... 7 - 4
7.3 Procedure for Communicating with an External Device ..... 7 - 5
7.3.1 Communications with a console ..... 7 - 5
7.3.2 Communicating with a terminal ..... 7 - 7
7.3.3 Output to a printer ..... 7 - 10
7.3.4 Communicating with other external devices ..... 7 - 12
7.4 Interrupt Processing from an External Device ..... 7 - 14

8. MULTITASKING ..... 8 - 1 ~ 8 - 20

8.1 Multitasking ..... 8 - 1
8.2 Synchronizing Execution (Event Processing) ..... 8 - 5
8.3 Using Devices (Resources) under Multitasking ..... 8 - 8
8.4 Starting Another Program from within a Program ..... 8 - 14
8.5 Transferring Data between Programs ..... 8 - 15
8.5.1 Common memory and internal devices ..... 8 - 15
8.5.2 The message port ..... 8 - 17

9. ERROR MANAGEMENT ..... 9 - 1 ~ 9 - 3

9.1 Defining Error Processing ..... 9 - 1
9.2 Error Type and How to Identify the Error Location ..... 9 - 2
9.3 Error Processing Difficulties ..... 9 - 3

10. DEBUGGING A PROGRAM ..... 10 - 1 ~ 10 - 3

10.1 Debugging Programs that Execute Simultaneously under Multitasking ..... 10 - 1
10.2 Commands Used for Debugging a Program ..... 10 - 1

11. COMMANDS AND FUNCTIONS ..... 11 - 1 ~ 11 - 356

ABS — Absolute value ..... 11 - 2
ASC — First character in string conversion to ASCII ..... 11 - 3
ATN — Returns the arctangent ..... 11 - 4
AUTO — Automatic line number display ..... 11 - 5
BEEP — Sound the beep ..... 11 - 6
BIN$ — Returns the binary representation of an integer as a character string ..... 11 - 7
BSWAP — Interchanges the byte position of two values ..... 11 - 9
CDBI — Conversion of double precision real number into 32-bit integer 11 - 11
CDBL — Conversion of integer single precision real number to double precision real number ..... 11 - 13

CHAIN	Program read, or execution after combination	11 - 15
CHR$	Character corresponding to the character code specified as a numeric expression	11 - 17
CIDB	Conversion of 32-bit integer into double precision real number	11 - 18
CINT	Conversion of signal precision real number, double precision real number into integer	11 - 20
CISN	Conversion of 32-bit integer into single precision real number	11 - 22
CLEAR	Initializes variables and sets up memory range	11 - 24
CLOSE	Completes a file I/O operation	11 - 25
CLS	Screen clear	11 - 26
COM ON/OFF/STOP	Enables/disables or stores input from the communication line	11 - 27
COMMON	Passes the variable to the program	11 - 28
CONSOLE	Setting the number of lines on console screen	11 - 29
CONT	Program restart after stop	11 - 30
COS	Cosine	11 - 31
CSNG	Conversion of integer, double precision real number into single precision real number	11 - 32
CSNI	Conversion of single precision real number to 32-bit integer	11 - 33
CVD	Conversion of character string created with MKD$ function back to double precision real number	11 - 35
CVDMBF	Conversion into IEEE format double precision internal representation	11 - 36
CVI	Conversion of character string created with MKIS$ function back to integer	11 - 37
CVS	Conversion of character string created with MKS$ function back to single precision real number	11 - 38
CVSMBF	Conversion into internal representation of floating point real numbers used by the A2A, A3A	11 - 39
DATA	Data setting for READ command	11 - 41
DATE$	Setting and read the year, month, day from PC CPU	11 - 43
DEFDBL	Definition of variables beginning with the characters in the specified range as double precision real variables	11 - 45
DEFFN	Definition of a user function	11 - 47
DEFINT	Definition of variables beginning with the characters in the specified range as integer variables	11 - 49
DEFSNG	Definition of variables beginning with the characters in the specified range as single precision real variables	11 - 51
DEFSTR	Definition of variables beginning with the characters in the specified range as character string variables	11 - 52
DEF ZEVENT	Definition of event with bit device	11 - 54
DELETE	Deletion of specified range of program	11 - 57
DIM	Array definition	11 - 58
END	Execution end and waiting for input, or not executing	11 - 60
EOF	Sequential file end detection	11 - 61
ERASE	Deletion of array from memory	11 - 62
ERL	Error-detected line number read	11 - 63
ERR	Error code read	11 - 64

ERROR	Error occurrence	11 - 65
EXP	Exponent	11 - 66
FIELD	Variable area allocation to random file	11 - 67
FILES	File name display	11 - 68
FIX	Truncating fraction part of numeric value	11 - 69
FOR ~ NEXT	Repeating execution of a sequence of commands a specified number of times	11 - 70
FORMAT	Initialization of file area of memory card	11 - 72
FRE	The size of remaining memory area in bytes	11 - 74
GET	Data read from random file into file buffer	11 - 75
GETMEM	Data read from buffer memory, common memory, or internal device (ED)	11 - 76
GOSUB RETURN	Branches to a subroutine	11 - 81
GOTO	Unconditional shift to specified line	11 - 83
HEX$	Conversion of integer data into hexadecimal stored as a character string	11 - 84
IF~GOTO~ELSE	Select a location to branch according to an expression result	11 - 86
IF~THEN~ELSE	Select a command to execute according to an expression result	11 - 88
INKEY$	Character of the key pressed	11 - 90
INPUT	Data input from the keyboard	11 - 92
INPUT$	Character string of a specified length which contains data from keyboard, file, or communication port	11 - 94
INPUT#	Data read from sequential file	11 - 97
INSTR	Search for a specified character string in a character string	11 - 98
INT	Conversion of numeric value into integer	11 - 100
KEY	Definition of character string to console function key	11 - 102
KEYLIST	Display character strings defined to console function	11 - 103
KILL	File deletion	11 - 104
LEFT$	Character string of specified length from left side of character string	11 - 106
LEN	Number of characters in character string	11 - 108
LET	Assignment of expression value to variable	11 - 109
LFILES	File name print-out	11 - 111
LINE INPUT	Store line of keyboard input to character variable	11 - 112
LINE INPUT #	Line read from sequential file to character variable	11 - 114
LIST	Program display	11 - 115
LLIST	Program print-out	11 - 117
LOAD	Program read	11 - 118
LOC	Current logical position in a file	11 - 119
LOCATE	Specification of display position on the console screen	11 - 120
LOF	Length of a file in record units	11 - 122
LOG	Value of natural logarithm	11 - 123
LPRINT	Data print-out	11 - 124
LPRINT USING	Data print-out in specified format	11 - 125
LSET	Data left-justify assignment from memory to random buffer file	11 - 126

MERGE	Conbination of program in current memory with program read	11 - 128
MID$(version 1)	Replacement of part of string to another string	11 - 130
MID$(version 2)	Fetching only specified part of character string	11 - 132
MKD$	Conversion of double precision type value into character string	11 - 133
MKDMBF$	Data conversion from IEEE internal representation to a string, can be converted into numeric value using CVD function	11 - 135
MKI$	Conversion of interger value into character string	11 - 136
MKS$	Conversion of single precision real number value into character string	11 - 138
MKSMBF$	Conversion of internal representation of floating point real number used by the A2A, A3A into string, can be converted into numeric value using CVS function	11 - 140
NAME	File rename	11 - 142
NEW	Program deletion from memory and variable initialization	11 - 144
OCT$	Conversion of integer data into octal character string	11 - 145
ON COM GOSUB	Defines the starting line of the control routine to which execution is branched when data is received from the communication line	11 - 147
ON ERROR GOTO	Specification of the first line of the error processing routine at error occurrence	11 - 149
ON~GOSUB	Branches execution to one of several specified subroutines based on a value	11 - 151
ON~GOTO	Branches execution to one of several specified line numbers based on a value	11 - 153
OPEN	File open to enable I/O processing	11 - 155
PCRD		11 - 157
PCRD -Procedure Code 1-	Read device memory data	11 - 159
PCRD -Procedure Code 2-	Monitor from a device that was recorded for monitoring with PCWT command	11 - 161
PCRD -Procedure Code 4-	Read data from the extension file register	11 - 163
PCRD -Procedure Code 5-	Monitor an extension file regester that was recorded for monitoring with PCWT command	11 - 165
PCRD -Procedure Code 7-	Read data from specified consecutive addresses within the extension file register	11 - 166
PCRD -Procedure Code 8-	Read sequence program	11 - 169
PCRD -Procedure Code 9-	Read microcomputer program	11 - 171
PCRD -Procedure Code 10-	Read comment data	11 - 173
PCRD -Procedure Code 11-	Read extension comment data	11 - 175
PCRD -Procedure Code 12-	Read the special function module's buffer memory	11 - 177
PCRD -Procedure Code 13-	Read PC CPU type	11 - 183
PCRD -Procedure Code 14-	Read parameter data	11 - 185
PCWT		11 - 188
PCWT -Procedure Code 1-	Write to device memory	11 - 190
PCWT -Procedure Code 2-	Record device memory for monitoring	11 - 192
PCWT -Procedure Code 3-	Random write to device memory	11 - 195
PCWT -Procedure Code 4-	Write to the extension file register	11 - 198
PCWT -Procedure Code 5-	Record the extension file register for monitoring	11 - 200
PCWT -Procedure Code 6-	Randam write to the extension file register	11 - 202

PCWT -Procedure Code 7-	Write to specified consecutive addresses within the extension file register	11 - 204
PCWT -Procedure Code 8-	Write sequence program	11 - 207
PCWT -Procedure Code 9-	Write microcomputer program	11 - 209
PCWT -Procedure Code 10-	Write comment data	11 - 211
PCWT -Procedure Code 11-	Write extension comment data	11 - 213
PCWT -Procedure Code 12-	Write to the special function module's buffer memory	11 - 215
PCWT -Procedure Code 14-	Write parameter data	11 - 221
PCWT -Procedure Code 15-	Analyze parameter data	11 - 224
PCWT -Procedure Code 16-	PC CPU remote STOP	11 - 225
PCWT -Procedure Code 17-	PC CPU remote RUN	11 - 226
PCWT -Procedure Code 20-	Issue PC CPU interrupt	11 - 227
PRINT	Data screen display	11 - 229
PRINT USING	Data screen display in specified format	11 - 231
PRINT#	Data write to sequential file	11 - 235
PRINT# USING	Data write to sequential file in specified format	11 - 236
PUT	Write to random file from random file buffer	11 - 237
PUTMEM	Data write to buffer memory, common memory,or internal device(ED)	11 - 238
RDSET	1-bit data read from specified bit	11 - 243
READ	Reads the values defined with DATA command and stores to variables	11 - 245
REM	Comment in the program	11 - 246
RENUM	Line renumbering	11 - 247
RESTORE	Specification of DATA defined by READ command	11 - 249
RESUME	Return from error management procedure	11 - 250
RIGHT$	Fetching character string of specified length from right	11 - 252
RND	Random number	11 - 253
ROT	Bit rotation	11 - 254
RSET	Moves data right-justified to random file buffer from memory	11 - 256
RUN(version 1)	Program execution	11 - 258
RUN(version 2)	Program read and execution	11 - 259
SAVE	Stores a program	11 - 260
SEARCH	Searches for a specified value within a specified array, and returns the position of the element where the value was found	11 - 262
SGN	Sign of a numeric value	11 - 264
SHA	Arithmetic shift on integer data	11 - 265
SHT	Logical shift on integer data	11 - 267
SIN	Sine	11 - 269
SPACE$	Blank character string of specified length	11 - 270
SPC	Specified number of blanks	11 - 271
SQR	Square root	11 - 272
STOP	Program stops temporarily, or waiting for command input	11 - 273
STR$	Conversion of numeric value assumed to be base 10 into character string	11 - 274
STRING$	Returns a string containing a specified number of a specified character	11 - 275

SYSTEM	Returns to system mode or main menu screen	11 - 277
SWAP	Exchanges the values of two variables	11 - 278
TAB	Advance the current display position to specified location	11 - 279
TAN	Tangent	11 - 280
TIME$	Setting the time of PC CPU, and reading the time	11 - 281
TROFF	Program trace reset	11 - 283
TRON	Program trace	11 - 284
VAL	Conversion of character string into numeric value	11 - 285
WHILE~WEND	Repeats the command as long as a specified condition is met	11 - 287
WIDTH	Setting the printer width	11 - 289
WTSET	Writes a zero or a one to a specified bit	11 - 290
ZBAS	BASIC task number of the program currently being created or executed	11 - 293
ZCLOSE	Communication channel close	11 - 294
ZCNTL		11 - 295
ZCNTL -Procedure Code 16-	Setting communication parameter	11 - 297
ZCNTL -Procedure Code 17-	Read communication parameter	11 - 297
ZCNTL -Procedure Code 18-	Setting communication control parameter	11 - 299
ZCNTL -Procedure Code 19-	Read communication control parameter	11 - 299
ZCNTL -Procedure Code 22-	Setting break character	11 - 302
ZCNTL -Procedure Code 23-	Read break character	11 - 302
ZCNTL -Procedure Code 32-	Setting the RS and ER signs ON/OFF	11 - 304
ZCNTL -Procedure Code 33-	Read the ON/OFF status of the CS, DR, RS, ER, and CD signals	11 - 306
ZCNTL -Procedure Code 64-	Read the cause of a receive error	11 - 308
ZCNTL -Procedure Code 80-	Read size of the buffer and number of stored characters	11 - 309b
ZCNTL -Procedure Code 128-	Read printer status	11 - 310
ZCNTL -Procedure Code 136-	Sending the initialize signal to the printer	11 - 312
ZEVENT	Enables or disables event signaling	11 - 313
ZIDV	Specification of input source from which input will be received for the INPUT and similar commands	11 - 314
ZLDV	Print port designation	11 - 315
ZMESSAGE	Definision of message port	11 - 316
ZMESSAGE CLOSE	Message port close	11 - 319
ZMESSAGE GET	Read message from message port	11 - 320
MESSAGE KILL	Message port deletion	11 - 323
ZMESSAGE OPEN	Message port open	11 - 324
ZMESSAGE PUT	Write message to message port	11 - 325
ZMOVE	Data transfer between variables	11 - 327
ZODV	Changes the output destination used by the PRINT and similar commands	11 - 332
ZOPEN	Communication channel open	11 - 334
ZRECEIVE	Recieving data from communication channel	11 - 336
ZRELEASE	Enables another program to use a resource that has been assigned a resource number	11 - 341

ZRESERVE — Makes it impossible for other programs to use a resource that has been assigned a resource number . . . . . 11 - 342
ZSEND — Sending data from communication channel . . . . . 11 - 345
ZSIGNAL — Event occurrence . . . . . 11 - 350
ZSTART — Specified program start . . . . . 11 - 351
ZURGENCY — Changes the priority of a program . . . . . 11 - 353
ZWAIT DELAY — Suspends program execution until a specified time period has elapsed . . . . . 11 - 354
ZWAIT EVENT — Suspends program execution until a specified event is signalled . . . . . 11 - 355

12. CREATING BASIC PROGRAMS USING A COMPILER . . . . . 12 - 1 ~ 12 - 20

12.1 Differences Between Compiler BASIC and Interpreter BASIC . . . . . 12 - 1
12.2 Flow When Creating a Program Using a Compiler . . . . . 12 - 2
12.3 Necessary Items For Compiling . . . . . 12 - 3
12.4 Registering Assemblers and Linkers to a Hard Disk . . . . . 12 - 3
12.5 Starting the Compiler . . . . . 12 - 4
12.6 Precautions When compiling . . . . . 12 - 6
12.7 Execution Using an AD51H-S3 . . . . . 12 - 7
12.8 Instructions and Functions . . . . . 12 - 8
12.8.1 Compilability of instructions and functions . . . . . 12 - 8
12.8.2 Different instruction and function specifications when using a compiler . . . . . 12 - 13

APPENDICES . . . . . APP - 1 ~ APP - 57

APPENDIX 1 FILE NAMES . . . . . APP - 1

1.1 Drive Numbers . . . . . APP - 1
1.2 System Names . . . . . APP - 2
1.3 File names . . . . . APP - 3
1.4 Wildcards . . . . . APP - 4
1.5 Precautions When Using Wildcards . . . . . APP - 5
1.6 How to Specify a File Name . . . . . APP - 5

APPENDIX 2 PRECAUTIONS WHEN DOING INTERRUPTION PROCESSING . . . . . APP - 6

APPENDIX 3 COMMANDS AND FUNCTIONS TO SWITCH PROGRAMS EXECUTED BY MULTITASKING . . . . . APP - 7

APPENDIX 4 CODE TABLES . . . . . APP - 8

4.1 Character Code Table . . . . . APP - 8
4.2 List of Control Key . . . . . APP - 10
4.3 Screen Display Control Codes When Using a General-Purpose Console . . . . . APP - 11
4.4 Lists of Error Messages and Error Codes . . . . . APP - 12
4.4.1 List of error messages . . . . . APP - 13
4.4.2 Table of system error codes . . . . . APP - 16
4.4.3 Error messages when compiling . . . . . APP - 24

APPENDIX 5 METHOD FOR REQUIRING TRIGONOMETRIC FUNCTIONS THAT ARE NOT PREPARED IN THE AD51H-BASIC ..... APP - 35

APPENDIX 6 RESERVED WORDS ..... APP - 36

APPENDIX 7 COMMUNICATIONS CONTROLS ..... APP - 38

7.1 DC/DC3 Controls ..... APP - 38
7.1.1 DC1/DC3 send controls enabled ..... APP - 38
7.1.2 DC1/DC3 receive controls enabled ..... APP - 40
7.2 Signal Controls ..... APP - 41
7.2.1 ER(DTR) control enabled ..... APP - 41
7.2.2 RS(RTS) control enabled ..... APP - 43
7.2.3 DR(DSR) control enabled ..... APP - 45
7.2.4 CS(CTS) control enabled ..... APP - 46
7.2.5 How to connect to an external device and when designating signal controls ..... APP - 47
7.3 Break Character Control ..... APP - 49
7.3.1 Example of break character specification ..... APP - 49
7.3.2 Received data flow when a break character is designated ..... APP - 51
7.4 Data Flow When an Error Occurs During Data Receive ..... APP - 53
7.4.1 Data flow when a transmission error occurs ..... APP - 53
7.4.2 Data flow when a full receive buffer error occurs ..... APP - 56
7.5 How to Clear Received Data Stored in the Receive Buffer ..... APP - 57
7.6 Communications Using 256 or More Bytes of Character Data ..... APP - 57

How to understand this manual

- To use interpreter BASIC, read Sections 1 to 11 and the Appendices. It is not necessary to read Section 12.
- To use compiler BASIC, read Section 12, and confirm the restricted item of compiler BASIC. And then, read Sections 1 to 11 and the Appendices.

Compiler BASIC

- When using compiler BASIC, each instruction has restricted items. Therefore, pay attention to these items. Section 12.8.2 gives details about restricted items.
- An assembler and a linker are necessary for compiler operations. But, since they are not contained in the SW1IX- AD51HPE software package, they must be purchased separately.

  Mitsubishi recommends the following:

  Microsoft Macro Assembler Ver.4.0 or Ver.5.1 (supporting AX)

  Microsoft Linker (see Section 12.3)
- Since compiler BASIC utilizes MS-DOS, the user should have a working knowledge of MS-DOS.

Module names

When "AD51H" and "AD51H-S3" are used in this manual, they indicate the following:

AD51H: The explanation involves items common to an AD51H and AD51H-S3
AD51H-S3: The explanation only involves an AD51H-S3.

1. GENERAL DESCRIPTION

This programming manual explains the "AD51H-BASIC" programming language (used for commands, functions, etc.) for use with an AD51H and AD51H-S3 intelligent communications module.

To use the AD51H-BASIC, consult the following manuals:

AD51H User's Manual

AD51H-S3 User's Manual

SW[ ]IX-AD51HPE AD51H/AD51H-S3 Software Package Operating Manual

- If a PC/AT is used (along with an AD51H) as the console (display terminal

• IBM is a registered trademark of the International Business Machines Corporation. • MS-DOS, Microsoft Macro Assembler are registered trademark of Microsoft Corporation.

1.1 Features

The AD51H-BASIC's features are shown below:

(1) Interpretative BASIC

Even for beginners who have little knowledge of its complicated internal structure, it is easy to create a system control program or a program to do data link with a computer.

(2) Multiple programs can be run at the same time using the multitasking executive system.

The AD51H supports multitasking.
Up to eight programs can be executed at the same time.
In this case, the user can benefit from the multitasking system because this system functions as if some programs are executed at the same time.

(3) Data communications can be executed between programs through such devices as the message port and the AD51H internal device

When some programs are executed at the same time using the multitasking system, data can be communicated as shown below:

1) Message port ..... 1:n data communications

2) AD51H internal device ..... 1:n data communications

3) Shared memory ..... 1:n data communications

(4) Synchronization between programs for each event

When some programs are executed at the same time using the multitask system, an event can be made to enter the wait state using the ZWAIT EVENT command or released from the wait state using the ZSIGNAL command.
Therefore, the programs can be synchronized.

(5) Same commands can be shared for communications with an external device

Data link with the CPU in which the AD51H is used or with an PC CPU via the MELSECNET is done by using the PCRD/PCWT command.
Data link between the AD51H interfaces and external devices is done by using the ZRECEIVE/ZSEND command.
Therefore, once a subroutine is created to do data link, this subroutine can be utilized for different programs by replacing the parameters required for each program.

1.2 Symbols Used in This Manual

The following symbols are used in this manual (especially in the sections explaining the format) for explanatory purposes:
These symbols must be fully understood before programming.
Since any of these symbols causes a syntax error in a program, these characters cannot be used in the program.
Other symbols and characters have different meanings. So, use them according to the explanation.

[ ] ..... indicates a part that can be omitted.

{ } ..... indicates a part that can be used or a part that can repeatedly be specified.

< > ..... indicates an item that can be specified by the user.

| ..... indicates selecting between items right to "|" or left to "|". When it is hard to find where the corresponding range starts and ends, that range is underlined.

Δ ..... indicates a part where one or more spaces should be placed.

H ..... indicates the preceding alphanumeric character string is an integral number expressed in hexadecimal.

[Ctrl] + [ ] ..... indicates a key pressed along with the [Ctrl] key.

[Enter] ..... indicates pressing the [Enter] key.

2. INTRODUCTION TO AD51H-BASIC

2.1 Preparations for Using AD51H-BASIC

When a program source is edited or the program is executed using AD51H-BASIC (hereafter called BASIC), set the AD51H to the edit mode, connect a console to the AD51H, and run BASIC.
The procedure is given below:

(1) Set the AD51H and connect it to the console

Section 4.2 in the AD51H/AD51H-S3 User's Manual (Hardware) gives details about this procedure.

(2) Prepare the console

1) If a PC/AT is used as a console, see Section 6.2 in the SW[ ]IX-AD51HPE Software Package Operating Manual, and execute the on-line programming items.

2) When a general-purpose terminal is used as the console, a software package is not necessary.

(3) As explained in the AD51H/AD51H-S3 User's Manual (Control Function), when the AD51H is in the system mode, giving a START command starts the editing of a BASIC program or executes the program.

```
S>START Δ [ ],[   ] [Enter]
```

(4) After the above operations are executed, "OK" appears on the console screen.

```
AD51-BASIC ON-LINE PROGRAMING Ver X.X
O K
■
```

POINT

Online programming mode refers to the state when the START command is given to the AD51H in the system mode.
In this mode, it is possible to create a BASIC program and to execute a single-task program.

(5) To return to the AD51H system mode after editing the BASIC program source or executing the program, either (a) execute the SYSTEM command, or (b) press the [Ctrl] and [D] keys at the same time.

```
S> ■
```

REMARKS

(1) When the SW[ ]IX-AD51HPE Software Package is used, the offline programming mode only used for editing a BASIC program source is available.
The SW[ ]IX-AD51HPE Software Package Operating Manual gives details about the offline programming mode.
Hereafter, only the online programming mode is discussed in this manual.

(2) When programs are executed using the multitasking system, settings for AD51H multitasking in the system mode should be done and the execution should be started.
Section 3.1 in the AD51H/AD51H-S3 User's Manual (Programming and Debugging) gives details about these procedures.

PRECAUTION

The AD51H contains two modes to execute a BASIC program:

(1) Edit mode (Online programming mode)

In this mode, it is possible to program online and to execute only the program being programmed.

The Multitasking system cannot work in the edit mode.

> Except for commands used for multitasking, this manual gives examples when a program source is edited or a program is executed in the edit mode.

REMARK

The offline program mode can only be used to create a BASIC program when the SW[ ]IX-AD51HPE software package is used.
The corresponding software package operating manual gives details about the offline programming mode.

(2) Execution mode

The multitasking system can be applied to a BASIC program that uses commands for multitasking.
A BASIC program cannot be created in this mode.

Switching between the edit and execution modes.
The AD51H/AD51H-S3 User's Manual gives details about these modes and how to switch them.

2.2 Direct Mode and Program Mode

- Direct mode

  When "OK" is displayed on the console screen, input a BASIC command, and press the [Enter] key.
  Then, the BASIC program is executed.
  This procedure is called an "execution in the direct mode".
  The result of the execution in the direct mode remains in the memory.
  The command statements, however, are cleared.
  All BASIC commands but DEF FN can be directly executed.

- Program mode

  Number program source lines, input a BASIC command, and press the [Enter] key.
  Then, the command is stored in the AD51H memory. A line consists of a number and a BASIC command.
  The BASIC program is a program includes one or more lines. The BASIC program can be called a program.
  A program is a set of commands to operate the computer.
  Programming refers to the stage to design and test a program.
  A program stored in memory can be executed by giving a BASIC RUN command.
  This procedure is called an "execution in the program mode".

2.3 Line Format

A line is the minimal unit in a program. It includes the line number, sentence, and the symbol ([Return]) indicating the end of the line.
A sentence is a collection of statements. ([Return] is input by pressing the [Enter] key during editing.)

The line format is shown below:

Sentence

number Δ [<Label>:] Statement {:<Statement>} [Return]

(1) Line number

<Line number> must be always input from the first column.
The line number must be expressed within the range of 1 to 65529 in integral numbers (decimal constants).
Place line numbers in ascending order in a program.
Line numbers are used as a reference for controlling the program and for searching a line during editing.

(2) Label

<Label> can be specified as a branchpoint of such commands as GOTO, GOSUB, and RESTORE or as data to be read next.
<Label> must be placed after the relevant line. When a line accompanied by abel is specified, that line can be specified using the Label instead of the line number.

- A label is a character string beginning with a asterisk (*) to identification. Not counting the asterisk, up to 15 alphanumeric characters and period can be included in the character string.
- A reserved word cannot be used as a label.
- A label name to be referenced must always come first in a line.
- When an command follows a label in a line, a colon (:) should be placed after the label.
- When the same label name is used repeatedly, only the first one will be referenced.
- In this case, doing so does not cause an error.

(3) Statement

<Statement> is an command that conforms to the BASIC grammar A <statement> is the minimal unit in a sentence.

(4) Colon (:)

A colon is placed between the two statements when two statements are integrated into one.
A sentence consisting of one statement is a simple sentence.
A sentence consisting of several statements (including colons) is a complex sentence (multistatement).

(5) Line length

The length of a line, including line numbers and spaces, must be 254 characters or less.
[Return] represents the end of a line.

2.4 Space and Key Word

The names of the commands and functions in a BASIC program are key words which have special meanings.
These names must not include any spaces (character of 20_H shown in Appendix 4.1).
To specify key words, variables, arrays, constants, and logical operators in a program, spaces, brackets, and numeric operators must be used in conformance with the BASIC grammar.
However, except the above case, a space can be used freely, and every space in a sentence is ignored.

Example

PΔRΔIΔNΔT	causes an error.
AΔ=ΔBΔ+ΔC	causesno error.
A=BΔANDΔC	means A contains B by C.
A=BANDC	means A contains BANDC.

REMARK

The name on the [Enter] key varies according to the console used.

2.5 Characters that Can be Used in a BASIC Program

The characters that can be used in a BASIC program are shown below:

- Upper case letters:ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Lower case letters: abcdefghijklmnopqrstuvwxyz
- Numerical characters:0123456789
- Special symbols:(space) !"#$%&'( )*+-, ./:;< >=?@[ ]\^{ } | -
  The Character Code List in Appendix 4.1 gives information about alphanumeric characters and special symbol codes.

The program commands are described using upper/lower case letters and symbols which conform to the BASIC grammar.
Other characters can be used in character data.
A lower case letter is regarded as the corresponding upper case letter.
Lower case letters composing a command are replaced by the corresponding upper case characters, and the command is stored in the main memory.
However, **character data** (such as quotes and file names) is stored to memory as it is.

The following special symbols are used.

Special Symbol	Name	Meaning
Δ	Space	Put between a command and a parameter
!	Exclamation mark	Represents single-precision
"	Quotation mark	"" indicates a character string.
#	Number	Represents a file number or multiple precision.
$	Dollar	Represents a character type
%	Percent	Represents an integer type
&	Ampersand	Used to specify a character format
'	Apostrophe	Represents a note
(	Left parenthesis	
)	Right parenthesis	
*	Asterisk	Multiplication sign
+	Plus sign	Plus sign, addition sign, and character string addition
,	Comma	Put between fields [Example] INPUT A, B PRINT X, Y, Z DATA 10, 13, 91, 4
-	Minus sign, hyphen	Minus sign, subtraction sign Used to specify a range of lines to which a LIST or DELETE command is applied. [Example] LIST 100 - 300 DELETE 1000 - 3000 DEFDBL A - H
.	Period	Used as a decimal point or as a pointer to manage line numbers in a BASIC program. The number of a line in which an error occurred during the execution of a program, the number of a line in which an END statement stopped the operation, or the number of a line where a new program line was inserted is set. [Example] LIST. AUTO.
/	Slash	Division sign
:	Colon	Partition between multistatements
;	Semicolon	Partition [Example] PRINT" Answer=" ; A PRINT X; Y; Z INPUT"A=" ; A
<	equality sign (less than)	

=	Equal sign	
>	Inequality sign (greater than)	
?	Question mark	Substitute for PRINT command [Example] ? A, B ? FRE (0) ? TIME$
[	Left large bracket	
\	Reversed slash	Integer division sign
]	Right large bracket	
^	Circumflex Power sign	
_	Underline	
‘	Grave accent	
[	Left medium bracket	
|	Vertical line	
]	Right medium bracket	
-	Overline	

2.6 Commands and Functions

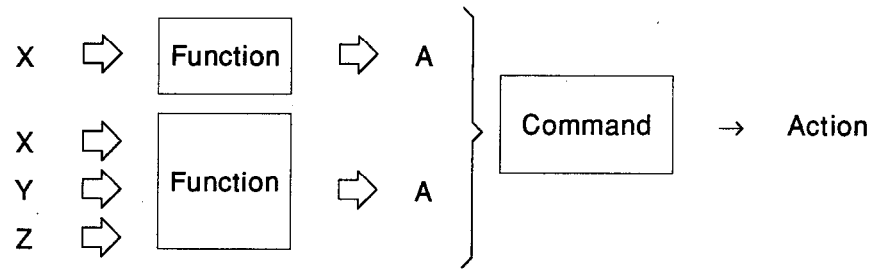
In a BASIC statement, use only commands and functions which conform to the BASIC grammar.
Commands and functions are explained below.

- Command

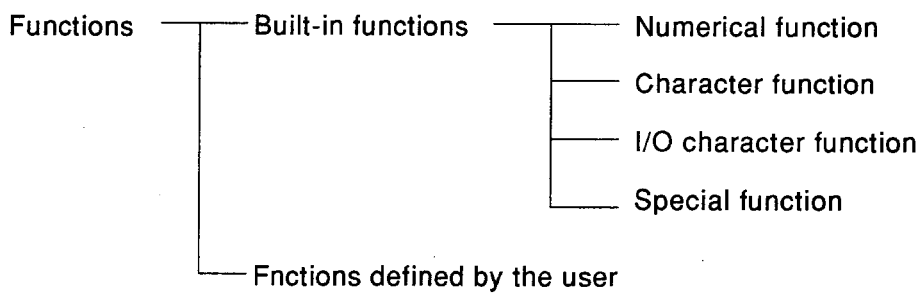
  Giving a BASIC command starts the corresponding operation.
  A PRINT command, for example, is given to display characters on the screen.

- Function

  The function represents correspondence between one or more given values and a given value.
  The function only corresponds those values. To use the result, it is necessary to give a command after executing the function.



Functions are categorized as follows:



The built-in functions are already supported to create a BASIC program.
The user can define a function by giving a DEFFN command as shown above.

2.7 Constants

A constant is an actual value used for the execution of a BASIC program. There are two kinds of constants: character string constants and numerical constants.

2.7.1 Character string constants

A character string constant is a group of quoted character strings, consisting of characters included in a set of up to 255 characters.

Example

"HELLO"
"$25, 000. 00"

POINT
A character string whose length is zero is called an empty character string. **In this manual, " " indicates an empty character string.**

2.7.2 Numerical constants

A numerical constant is a positive or negative number.
A numerical constant cannot contain a comma.
There are five kinds of numerical constant as shown below:

(1) Integer constant

This is an integer ranging from -32768 to 32767 or a number accompanied by %.
Decimal points cannot be used.

Example

100
+123
- 32768
32767%

(2) Fixed-point constant

This is a positive or negative real number (including a decimal point).
An integer which is not an integer constant is a fixed-point constant.

Example

100.00
-123.21

(3) Floating-point constant

This is an exponentially expressed positive or negative number.
The floating-point constant consists of an integer constant or fixed-point constant (fixed-point part), followed by "E" and an integer (characteristic).
Floating-point constants range from 10^{-38} to 10^{+38}

Example

235.988E-7 ..... 0.0000235988
2359E6 2359000000

(A double-precision floating-point constant uses "D" for "E".)

(4) Hexadecimal constant

This is a hexadecimal number (0 to 9, A to F) beginning with "&H".

Example

&H76 118
&H32F 815

(5) Octal constant

This is an octal number (0 to 7) beginning with "&O" or "&".

Example

&0347 231
&1234 668

2.7.3 Single-precision and double-precision constants

Fixed-point and floating-point constants are divided into single-precision and double precision constants respectively.
A single-precision constant is stored in memory as a value of 16 significant digits.

(1) A single-precision constant is one of the following numerical constants:

(a) A constant that has seven or less significant digits.

(b) An exponential number accompanied by "E".

(c) A constant ending with an exclamation mark (!).

(2) A double-precision constant is one of the following numerical constants:

(a) A constant that has eight or more significant digits.

(b) An exponential number accompanied by "D".

(c) A constant ending with "#".

Example

Single-precision constant	Double-precision constant
46. 8	345692811
–7. 09E –06	-1.09432D–06
3489.0	3489.0#
22.5!	7654321.1234

2.8 Variables

A variable is used in a BASIC program as a value.
Its value is either defined by the programmer or obtained from a calculation.
Before definition, the value of a numerical variable is zero, and the value of a character variable is empty.

2.8.1 Variable names and type-declaration characters

A variable should be named according to the rules shown below:

- The length of a BASIC variable name can be up to 15 characters.
- A variable name consists of alphanumeric characters (a period can also be used).
- A variable name must begin with an alphabetic character.
- Special characters such as %, !, #, and $ can be used as type-declaration characters.
- A reserved word cannot be used as a variable name, unless the variable name includes a reserved word.
  A reserved word is a command, function name, or operator name.
- When a variable name begins with "FN", that is regarded as the calling name of a function defined by the user.
- A variable represents a numerical value or character string.
- A character string variable name must end with a "$".
  For example, in A$ = "SALESREPORT", the dollar sign is a variable type-declaration character, declaring that this name represents a string character.
- A numeric variable name can be used for a double-precision constant or an integer.
- The following characters can be used as variable type-declaration characters:

%	Integer variable
!	Single-precision variable
#	Double-precision variable
$	Character string variable

When a numerical variable does not include a variable type-declaration character, that variable is regarded as a single-precision number.
Variable name examples are shown below:

Example

PI# ..... Used for a double-precision variable

MINIMUM! ..... Used for a single-precision variable

LIMIT% ..... Used for an integer variable

N$ ..... Used for a character string variable

ABC ..... Represents a single-precision variable

It is also possible to use commands such as DEFINT, DEFSTR, DEFSNG, and DEFDBL for variable-type declaration.

2.8.2 Array variables

- An array is a collection of values that can be referred to as a name.
- An array element is called an array variable name. BASIC commands and functions can handle array variables as variables.
- An array consists of some elements, and some dimensions, having a value type.
- Value types can be specified in the same way as variable names.
  Normally, it is necessary to declare the number of dimensions and the number of elements.
- The DIM command is used for an array. For example,

  10 DIM A$ (3) ..... Represents a declaration of a one-dimensional array named A$ (consisting of character string values).

  20 DIM B (3,2) ..... Represents a declaration of a two-dimensional array named B (consisting of numerical values).

- An array has as many subscripts as dimensions.
  A subscript refers to a location in an array. A subscript ranges from 0 to 32767.
  Elements of the above-mentioned A$ and B arrays, for example, can be referred to as shown below:

Array A$

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

Array B

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

- There can be up to 255 dimensions in an array.
- When a array is referred to before the declaration by the DIM command, the array is regarded as consisting of up to ten arrays.
  For example,

  30 C (3) =200

  When the command is executed before the declaration of C, this array is regarded as a one-dimensional array consisting of elements C(0) to C(10). However, such an implicit type-declaration can be applied only to an array that has two dimensions or less.

POINT

Some AD51H-BASIC commands do not support an implicit type-declaration of an array.
Mitsubishi recommends that an array should be used after declaring the array using the DIM command.

2.8.3 Special variables (how to use B@ and W@)

- These variables are used to read/write from/to each device of the AD51H using a BASIC program.
- There are four kinds of special variable as shown below:

  B@ (EM expression) .....Reads/writes from/to a bit device in the AD51H.

  W@ (ED expression) ..... Reads/writes from/to a word device in the AD51H.

  B@ (X expression) ..... Writes to general-purpose input X of the AD51H.

  W@ (Y expression) ..... Reads from general-purpose output Y of the AD51H.

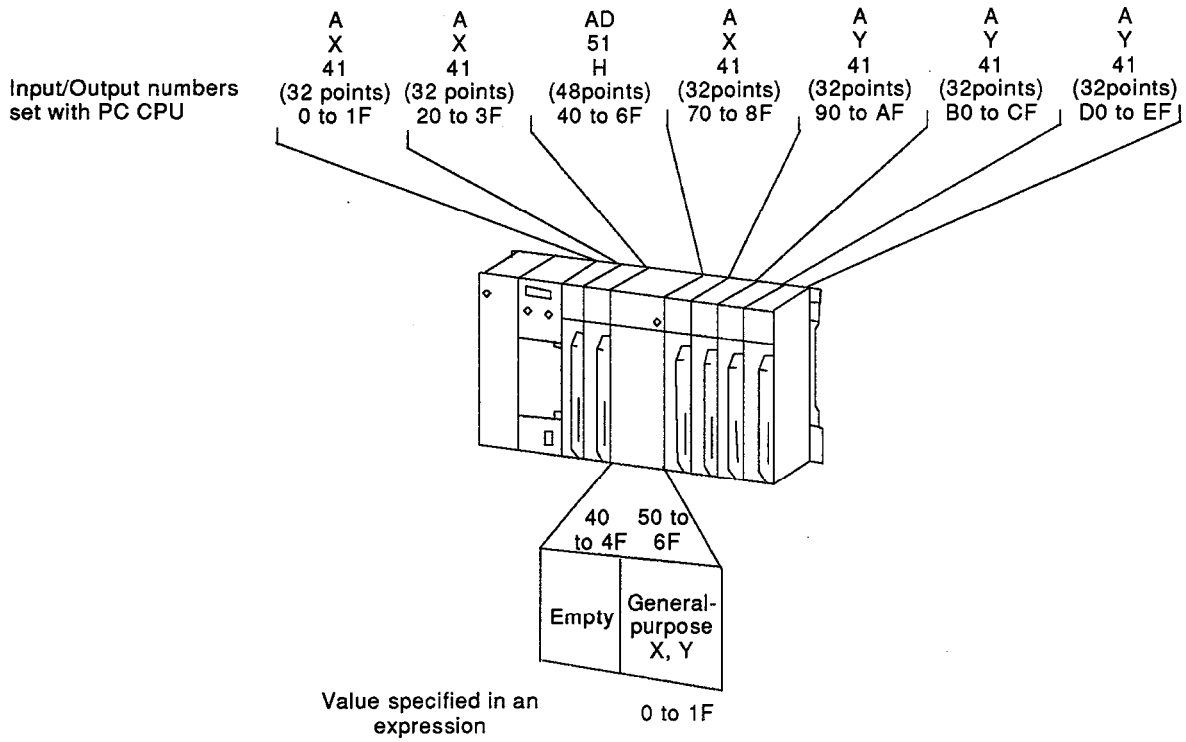
The AD51H/AD51H-S3 User's Manual (Control Function) gives details about internal devices EM and ED, and general-purpose inputs and outputs.
Data can be read/written from/to both special devices EM and ED.

(1) Bit device read/write

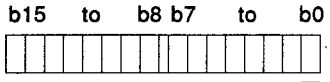
B@ (EM expression) ..... Specify 0 to 1023 in an expression.

B@ (X, expression) / B@ (Y, expression) ..... Specify 0 to $1F_H$ in an expression. Since the value is specified in hexadecimal, the value must begin with "&H".

- An expression includes a device number.
  When specifying general-purpose input and output, use 0 to $1F_H$ independently of system input and output numbers.



- When the read data is "1", the corresponding device goes ON.
  When the read data is "0", the corresponding device goes OFF.
- When data is written, only the lowest bit in the specified value or a value (stored in a specified variable) becomes valid.



If this is "1", the corresponding device goes ON.
If this is "0", the corresponding device

Therefore, set data using an integer constant or integer variable when the data is written.

- If a general-purpose input X is specified, data can only be written.
  If a general-purpose output Y is specified, data can only be read.

Example

```
A=B@ (EM,100)
A%=B@ (Y,&H1B)
B@ (EM,200) =A%
B@ (X,&HA) =A%
```

(2) Word device read/write

W@ (ED, expression) ..... Specify 0 to 1023 in an expression.

- The expression includes a device number.
- When written data is neither an integer constant nor an integer variable, the value is converted into an integer value and written.

Example

A = W@ (ED, 50) ..... Reads the data written to ED50 and assigns the data to single-precision variable A.

A% = W@ (ED, 50) ..... Reads the data written to ED50 and assigns the data to integer variable of A%.

W@ (ED, 80) = 1234 ..... Writes "1234" to ED80.

W@ (ED, 80) = A ..... Writes the single-precision real number A to ED80.

- When 32-bit data is read, read the data using an array variable of integer type, convert the data using the CIDB and CISN commands to the 32-bit data.

Example

```
100 DIM A%(1)
110 A%(0)=W@ (ED,100)
120 A%(1)=W@ (ED,101)
```

.... Reads the 32-bit data stored in ED100 and ED101 using A%(0) and A%(1) as 16-bit data.

130 B#=CIDB (A%(0)) ..... Converts the data in A%(0) and A%(1) to a 32-bit data and assigns that data to B#.

- When 32-bit data is written, the data is set to integer-type array variables and written to each device.

Example

```
100 DIM A%(1)
110 CDBI 123456!, A%(0)
120 W@ (ED, 102)=A%(0)
130 W@ (ED, 103)=A%(1)
```

110 CDBI 123456!, A%(0) ..... Handles "123456" as 32-bit data, storing the higher 16 bits of data to A%(0) and the lower 16 bits to A%(1).

120 W@ (ED, 102)=A%(0)
130 W@ (ED, 103)=A%(1) ..... Writes the higher 16 bits in A%(1) to ED103, and the lower 16 bits in A%(0) to ED102.

POINT

It is also possible to read/write from/to internal word device ED using the PUTMEM and GETMEM commands as well as special variables.
Section 8.5.1 describes the PUTMEM and GETMEM commands.

2.9 Type Conversion

The BASIC automatically converts a type of numerical to another type as follows:

(1) When a numerical constant is assigned to a different numerical variable, the value of the constant is converted to a declared type and stored. (If a value is assigned to a character string variable, or vice versa, a "Type mismatch" error occurs.)

Example

```
1 0    A%=23. 42
2 0    PRINT A%
RUN
 23
OK
■
```

(2) If an expression includes different types of numerical constants and numerical variables, the calculation is obtained as shown below:

- When two values are used for the calculation:

Value 1	Value 2		Calculation
Integer	Integer	→	Integer
Integer	Single-precision real number	→	Single-precision real number
Integer	Double-precision real number	→	Double-precision real number
Single-precision real number	Single-precision real number	→	Single-precision real number
Single-precision real number	Double-precision real number	→	Double-precision real number
Double-precision real number	Double-precision real number	→	Double-precision real number

(The same results will be obtained even if value 2 comes first.)

Example

```
10 D#=6#/7
20 PRINT D#
RUN
 .85714285714218571
OK
■
```

Since the double-precision number is divided by the single-precision number, the calculation is made in double precision.
The result is assigned as a double-precision value to D#.

(3) When a logic operation is done, the numerical constant and numerical value in the expression are converted and calculated.
The value must range from -32768 to 32767. If a value is outside the range, an "Overflow" error occurs.

(4) When a fixed-point numerical value is converted into an integer, numbers after the decimal are rounded off.

Example

```
10 C%=55. 88
20 PRINT C%
RUN
 .55
OK
■
```

(5) When a single-precision value is assigned to a double-precision variable, only the first seven digits of the converted value that has been rounded off become valid.
This is because only eight-digit precision can be obtained from a single-precision value.

Example

```
10 A=2. 04
20 B#=A
30 PRINT A;B#
RUN
 .2. 04 2. 039999961853027
OK
■
```

2.10 Expressions and Operators

The expression is to combine constants and variables by using operators in order to find a character string constant, numerical constant, variable, function, or value.
There are three kinds of operators.

(1) **Arithmetic operator**

(2) **Comparison operator**

(3) **Logical operator**

2.10.1 Arithmetic operators

If an expression includes some arithmetic operators, the operator precedence is shown below:

Operator	Operation	Example
^	Exponentiate	**X^Y**
-	Sign handling	**-Y**
*, /	Multiplication, floating-point division	**X*Y, X/Y**
+, -	Addition, subtraction	**X+Y**

When an operation order is changed, use parentheses. In an expression including parentheses, the operation in the parentheses is handled first.
When operators appear in a row, these must be partitioned by using parentheses.
In an arithmetic operation, a double-precision real number calculation can also be made.
However, in an exponential operation, all double-precision real numbers are converted into single-precision real numbers and the calculation is made.
The correspondence of algebraic expressions to BASIC expressions is shown below:

Algebraic expressions	BASIC expressions
$X+2Y$	**X+Y*2**
$X-\frac{Y}{Z}$	**X-Y/Z**
$\frac{X\ Y}{Z}$	**X*Y/Z**
$\frac{X+Y}{Z}$	**(X+Y)/Z**
$(X^2)^y$	**(X^2)^Y**
X^{Y^Z}	**(X^2)^Y**
$X\ (-Y)$	**X*(-Y)**

POINT

Signs for multiplication, division, and exponentiate in a BASIC expression are different from those in an algebraic expression.

$\times \rightarrow$ *
$\div \rightarrow$ /
$[\ ]^n \rightarrow [\ ]$^n

- Integer division and residue operation

  Integer division is expressed by "\". The decimal parts of the divisor, dividend, and quotient are rounded off before the operation.

  Example

```
PRINT 10\4
2
OK
PRINT 25. 68\6. 99 ............ Dividend: 25, Divisor:
6, Quotient: 4
4
OK
■
```

  A residue operation is expressed by the MOD operator, giving a residue produced by doing division as an integer.

  Example

```
PRINT 15. 4 MOD 4 ............ 15 divided by 4 produces a residue
3                               of 3.
OK
PRINT 25. 68 MOD 6. 9 ........ 25 divided by 6 produces a residue
1                               of 1.
OK
■
```

- Division when an overflow occurs and when a divisor is "0"

  During operation, when a divisor becomes "0", a "Division by zero" error occurs.
  Also, during exponential operation, when "0" is negatively exponentiated, a "Division by zero" error occurs.
  An overflow causes the "Overflow" error.

- The above-mentioned operators used along with other operators

  In an expression, a logical operator can be used as well as an arithmetic operator.
  In an expression including a comparison operator, when its result is true, the calculation is made using its value of "-1";
  when its result is false, the calculation is made using its value of "0".

  Example

  (A=B) * (A>C)

  (A AND MASK) /2

2.10.2 Comparison operators

A comparison operator is used for comparing a value with another one.
The comparison result is obtained as true (-1) or false (0).
This result is used to determine in which direction a program flows.

Operator	Operation	Example
=	Equal	**X=Y**
< >, ><	Not equal	**X< >Y,X><Y**
<	Less than	**X<Y**
>	Greater than	**X>Y**
<=, =<	Equal or less than	**X<=Y,X=<Y**
>=, =>	Equal or greater than	**X>=Y,X=>Y**

POINT

The equal sign is also used when a value is assigned to a variable.

When an expression includes both comparison and arithmetic operators, the arithmetic operation is always handled first, followed by the comparison operation.

Example

X+Y< (T-1) /Z ..... If X + Y is less than (T - 1)/Z, it will be true.

SIN (X) <0 ..... If SIN(X) is less than 0, it will be true.

2.10.3 Logical operator

The logical operator is used for handling a bit and doing a Boolean operation.
The logical operator gives a value of true (other than 0) or false (0).
In an expression, the logical operation follows arithmetic and comparison operations.
The results of logical operations are shown below:
Operators divided into six groups according to operator precedence.

1) NOT; Negation

X	NOT X
1	0
0	1

2) AND; Logical product

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

3) OR; Logical addition

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

4) XOR; Exclusive logical product

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

5) IMP; Inclusive

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

6) EQV; Equivalence

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

The logical operator (as well as the comparison operator) is used to determine program flow by comparing two or more values, giving a value of true or false.

Example

D<2 0 AND F<4 ..... If D is less than 200 and F is less than 4, it will be true.

I>1 OR K<0 ..... If I is greater than 10 or K is less than 0, it will be true.

NOT P ..... If P is equal to 0, it will be true.

PRECAUTIONS

(1) The logical operator converts a numerical constant and numerical variable into an integer ranging from -32768 to 32767, starting the operation.
(If the value does not range from -32768 to 32767, an overflow error will occur.)

(2) A logical operation works in units of bits, giving resultant bits corresponding to the two integer numbers.
For example, the AND operator is used for masking all bits but one. By using the OR operator, contents for two bites can be combined, producing a binary number.
The following shows how the logical operator actually works:

Example

63 AND 16 ..... 16 63 is 111111 in binary. 16 is 10000 in binary. Therefore, 63 AND 16 is 16.

```
      63 .... 0000 0000 0011 1111
AND   16 .... 0000 0000 0001 0000
             ------------------------
             0000 0000 0001 0000 .... 16
```

15 AND 14 ..... 14 15 is 1111 in binary. 14 is 1110 in binary. Therefore, 15 AND 14 is 14.

-1 AND 8 ..... 8 -1 is 1111111111111111 in binary. 8 is 1000 in binary. Therefore, -1 AND 8 is 8.

4 OR 2 ..... 6 4 is 100 in binary. 2 is 10 in binary. Therefore, 4 OR 2 is 6.

```
      4 .... 0000 0000 0000 0100
OR    2 .... 0000 0000 0000 0010
             ------------------------
             0000 0000 0000 0110 .... 6
```

10 OR 10 ..... 10 10 is 1010 in binary. Therefore, 1010 OR 1010 is 1010 (or 10).

-1 OR -2 ..... -1 -1 is 1111111111111111 in binary. -2 is 1111111111111110 in binary. Therefore, -1 OR -2 is -1.

NOT X ..... - (X+1) A complement of two corresponding to a given integer is the integer whose bits are inverted + 1.

```
            3 .... 0000 0000 0000 0011
Inverts bits .... 1111 1111 1111 1100
1 is added .....                    1
           ------------------------------
           -3 .... 1111 1111 1111 1101 ....
```

Complement of two

2.11 Character String Operations

Character strings can be combined using the + operator.

Example

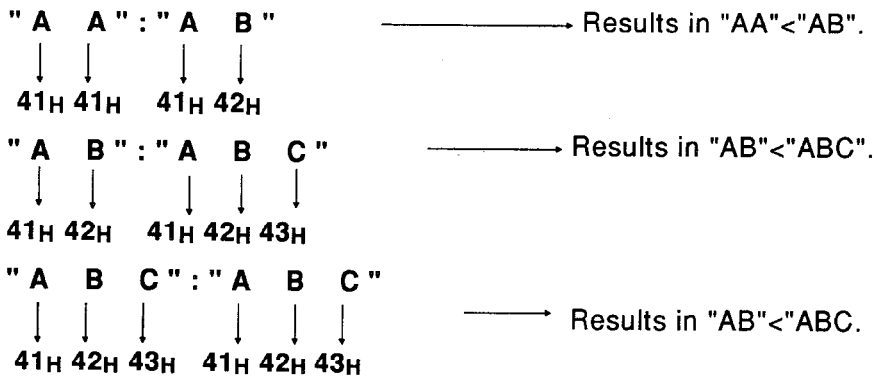
```
10    A$="FILE" :B$="NAME"
20    PRINT A$+B$
30    PRINT  "NEW"+A$+B$
RUN
FILENAME
NEWFILENAME
OK
■
```

By using the following comparison operators, a comparison between character strings can be made.

= < > < > < = >=

Comparisons of character strings are made character by character by handling characters as values, starting with the first characters in the string. (Appen4.1)
When each character in one string matches the corresponding characters in another string, that means these strings are equal.
Even if one character is different than its corresponding one in another string, the lesser string (lesser in terms of number of bites) is decided by the comparison between those characters.
Also, during a comparison, if the end of string is reached in one string, the shorter string is considered as less than the other.
Spaces at the beginning or end of a string are also compared.

Example



As shown above, through a character string comparison, character strings can be put in order alphabetically.
To compare character strings, all compared strings must be quoted (").

2.12 Operation Precedence

The operation precedence is shown below:
If two operations have the same priority, the one located to the left in the expression is handled first.

Operation in parentheses

Function

Exponent (Power) ^

Minus sign (-)

*, /

\

MOD

+,-

Comparison operators (<, >, =, etc.)

NOT

AND

OR

XOR

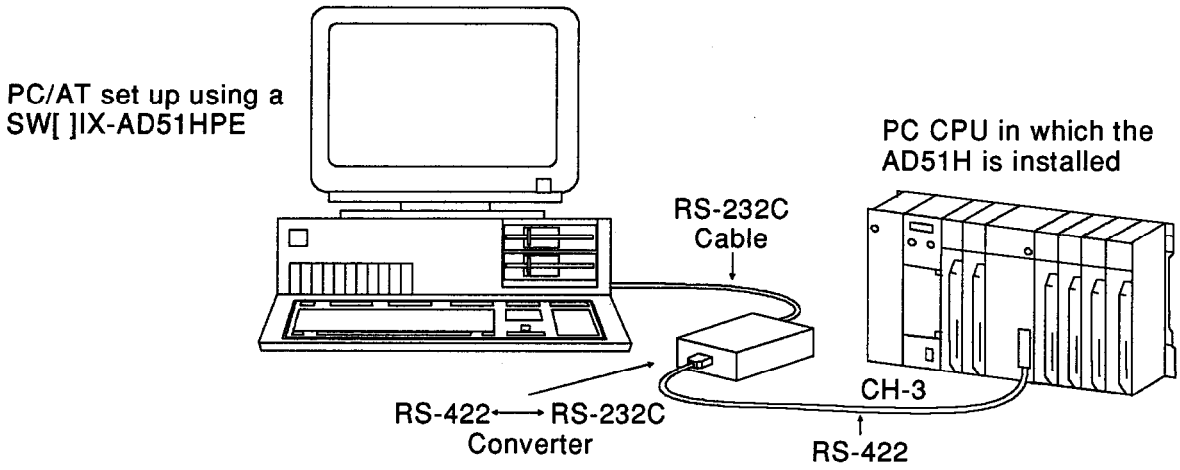
IMP

EQV

3. HOW TO CREATE AND EXECUTE A PROGRAM

This section describes how to input an AD51H-BASIC program, how to execute the program, and how to use BASIC commands.

Creating an actual program and seeing if the appropriate result can be obtained.
This section assumes that the following system is used:



PC/AT as a console is set up using a SW[ ]IX-AD51HPE

The AD51H and PC/AT must be set as shown below:

AD51H: Set the console in the edit mode to RS-422 interface.

Reference manual

- AD51H/AD51H-S3 User's Manual (Hardware)

PC/AT: Start up the SW[ ]IX-AD51HPE system. Then select the online programming so that the BASIC can be used in the AD51H system mode.

Reference manuals

- SW[ ]IX-AD51HPE Software Package Operating Manual (gives details about the PC/AT startup)
- AD51H/AD51H-S3 User's Manual (Programming and Debugging)
  (gives details about the AD51H system mode)

POINTS

(1) The online programming mode (that is supported by the SW[ ] IX-AD51HPE) can only be used to input a BASIC program. The appropriate software package operating manual gives details about the online programming mode.

(2) If a machine other than PC/AT is used as the console, the corresponding keys are:

[Enter]	Carriage return key (corresponds to such keys as[CR])
[Ctrl]	Control key(corresponds to a key such as [CTRL] and [CNTL])
[Insert]	Insert key (corresponds to a key such as [INS])
[Delete]	Delete key(corresponds to a key such as [DEL])
[Back Space]	Back Space key(corresponds to a key such as [BS])

3.1 Program Input

To input a program, first set up the console.
Start up the system so that AD51H-BASIC can be used in the console.
Section 2.1 gives general information about how to start up the system. For details, see the AD51H/AD51H-S3 User's Manual (Control Function).
Make sure that the console shows the following:

```
OK
■
```

This blinking mark is called a cursor.

"OK" indicates that the AD51H-BASIC is waiting for your command.
Now, input as shown below:

NEW [Enter]

The NEW command is used for deleting all programs from the memory. If a new program is input without deleting the old ones, these programs will be mixed. So, always make sure to execute a NEW command before any new program is input.

Now, the preparations for program input have been made. Input the following program.

```
10 REM Compound interest calculation
20 INPUT "Principal sum =";M
30 INPUT "Interest rate =";R
40 INPUT "Term =";K
50 RR=R/100
60 G=M*(1+RR)^K
70 PRINT "Principal rate =";INT(G);"Yen"
80 END
```

When giving an command in a program, its line must have a number. In this case, it is convenient to use the [AUTO] command to number the lines.
Input the command as follows:

AUTO [Enter]

A line number appears as shown below:

```
10Δ■
```

REMARK

If the [Enter] key is pressed and "Syntax error" is displayed on the screen, the input command contains a syntax error. Input the correct command again.
If a wrong key is pressed, delete it by using the [Back Space] key before pressing the [Enter] key.

Input line 10 as shown in the program example, and press the [Enter] key.

REM Compound interest calculation [Enter]

Line 20 is displayed on the screen. Thereafter, repeat this step until line 70 is displayed.
The screen is in the following state after line 70 is displayed.

```
70   END
80   ■
```

Since program input has now been completed, give the AUTO command to complete the line number display.
Press the following keys:

[Ctrl] + [C] ..... Should be pressed at the same time.

"OK" reappears on the display.

Using the LIST command, make sure that the program has been correctly input. Input as follows.

LIST [Enter]

If the displayed contents are the same as the program on the previous page, the input program is correct.

```
LIST
10 REM Compound interest calculation
20 INPUT "Principal sum =";M
30 INPUT "Interest rate =";R
40 INPUT "Term =";K
50 RR=R/100
60 G=M*(1+RR)^K
70 PRINT "Principal sum =";INT(G);"Yen"
80 END
OK
■
```

REMARK

If the program does not operate as shown above, correct the wrong line and change the incorrectly input line (and all subsequent lines) while the screen is in the OK state, and press the [Enter] key. Retry the LIST command, and the corrected contents are displayed.
Section 3.2.3 tells how to easily correct a wrong line.

3.2 Executing and Correcting a Program

3.2.1 Executing a program

Run the program input according to Section 3.1. In the OK state of the console, input as follows:

RUN [Enter]

The RUN command is used for executing a program stored in memory. The screen enters the following state:

```
OK
RUN
Principal sum=? ■
```

The program shown in Section 3.1 is used to make a calculation of compound interest. "Principal sum = ?" is an INPUT command on line 20. This command prompts the operator to input the principal sum. Input as follows:

10000 [Enter]

This means that the principal sum is 10,000 yen. Thereafter, the console displays "Interest rate = ?", followed by "Term = ?". Do the following: 5 [Enter] , 3 [Enter] This means that the interest rate is five percent and the term is three years. Completing the term input displays the calculation, completing the program.

```
Principal sum = 11576 yen
OK
■
```

(Therefore, investing 10,000 yen for three years at an annual compound interest rate of five percent yields a total of 11,576 yen.)

3.2.2 If an error occurs

When a program is executed by giving the RUN command, if the program contains an error, the following message will appear and the execution will be canceled.

```
xxxxxxx in NNNN
OK
■
```

This signals that an error prevents the program from operating normally. The "XXXXXX" part in the message indicates what kind of error occurred. The NNNN part indicates in which line the error occurred.
Appendix 4.4 tells what the error messages stand for.

3.2.3 Correcting a program

If a program (a) contains an error, or (b) is modified, the program must be corrected. There are two ways to correct a program.

(1) Correction line by line

Correct the program line by line.
A new line can be inserted or deleted as well.

- When a new line is inserted, specify its number, input an command, and press the [Enter] key.
- In this case, if the specified number already exists in that program, the new contents will be overwritten on the old one without displaying a confirmation dialog box.
- To place a new line at the end of program, specify a number higher than the final line number in the program, input an command, and press the [Enter] key.

```
 ≀
120   PRINT "END"
```

⇨ Specifying a number higher than 120 enables a new line to be placed at the end of the program.

Example

130 PRINT "BYE!!" [Enter]

- When a new line is inserted, specify a new number ranging from the starting number to the final number of the program, and press the [Enter] key.

```
 ≀
10 INPUT "PRODUCTION MARK";A
20 IF A=0 GOTO 120
 ≀
```

⇨ Specifying a new number between 10 and 20 enables a new line to be inserted.

Example

15 IF A,0 GOTO 10 [Enter]

- When a line is deleted, input only the line number, then press the [Enter] key.

```
 ≀
30 PRINT "TIME";TIME$
 ≀
```

⇨ Input 30, then press the [Enter] key to delete line 30.

- When continuous lines are deleted in batch, use the DELETE command.

Example

DELETE 600 ..... Deletes line 600

DELETE 300-400 ..... Deletes lines 300 to 400.

DELETE -500 ..... Deletes lines 1 to 500.

A corrected program can be confirmed by using the LIST command.
The line numbers of the displayed program are renumbered.

- Press the [Ctrl] + [S] keys to interrupt a LIST display.
- Press any key other than the [Ctrl] + [C] keys to resume the LIST display.
- Press the [Ctrl] + [C] keys to cancel the LIST display.

(2) Correction using the screen editor

Call the program, then move the cursor to the line to be corrected.
For correcting programs, the screen editor is more user-friendly than the line editor.

1) Correction using the screen editor is shown below:

The LIST command can be specified as follows:

LIST ..... Displays all the lines in a program.

LIST 300-400 .. Displays lines 300 to 400 in a program.

LIST -500 ..... Displays lines 1 to 500 in a program.

LIST 600- ..... Displays lines 600 and after in a program.

The LIST command can be canceled to correct a program. In this case, press the [Ctrl] + [C] keys.

2) To move the cursor, press the following keys:

[↑] ..... Moves the cursor to the previous line.

[↓] ..... Moves the cursor to the next line.

[→] ..... Moves the cursor right by one character.

[←] ..... Moves the cursor left by one character.

3) Correct a program using the following keys:

[Insert] ..... Places a space before the spot where the cursor is located.

[Delete] / [Back Space] ....deleted; the following part moves one character to the left. The character to the left of the cursor is

[Ctrl] + [E] ..... Deletes all characters from where the cursor is located to the end of the line.

4) After the correction is completed, press the [Enter] key.

POINTS

(1) If the [Enter] is not pressed, the changed program will not be stored in memory. Be sure to press the [Enter] key to store the changed program.

(2) Appendix gives details about other keys used for correcting a program.

(3) Correction examples using the screen editor

1) Overwrite correction ... Changes the command on line 20 to B=7.

1 Press the [↑] key to move the cursor to line 20.

```
LIST
10  A=2
20  B=[3]
30  C=A+B
40  PRINT C
50  END
OK
```

2 Press the [→] key to move the cursor to "3".

3 Input ["7"], and press the [Enter] key. The correction is now completed.

```
LIST
10  A=2
20  B=7
30  C=A+B
40  PRINT C
50  END
OK
```

2) Insert ... Changes the command on line 30 to C=100+A+B

1 Move the cursor to "A" on line 30.

```
LIST
10  A=2
20  B=7
30  C=[A]+B
40  PRINT C
50  END
OK
```

2 Press the [Insert] key four times, providing the necessary spaces.

3 Input "100+", and press the [Enter] key. The insertion is now completed.

```
LIST
10  A=2
20  B=7
30  C=100+A+B
40  PRINT C
50  END
OK
```

3) Delete ... Changes the command on line 30to C=A.

1 Move the cursor to "A" on line 30.

```
LIST
10  A=2
20  B=7
30  C=100+A+B
40  PRINT C
50  END
OK
```

2 Press the [Delete] or [Back Space] four times, deleting "100+".

```
LIST
10  A=2
20  B=7
30  C=A+B
40  PRINT C
50  END
OK
```

3 Move the cursor to "+" on line 30.

4 Press the [Crtl] + [E] keys, deleting "+B", and press the [Enter] key to complete the correction.

```
LIST
10  A=2
20  B=7
30  C=A
40  PRINT C
50  END
OK
```

4) Line number correction: Carefully correct a line number. Change the line number from 10 to 15.

1 Move the cursor to line 10.

```
LIST
↑10  A=10*5
↑20  PRINT A
↑OK
■
```

```
LIST
15  A=10*5
20  PRINT A
OK
■
```

2 Input "15", and press the [Enter] key.

The screen shows "OK". However, check the program by giving a LIST command (which will show that line 10 still exists).

```
LIST
15  A=10*5
↓20  PRINT A
↓OK
LIST
10  A=10*5
15  A=10*5
20  PRINT A
OK
10
LIST
15  A=10x5
20  PRINT A
OK
■
```

This is because no command has been given to delete line 10.

3 Input "10", and press the [Enter] key.(See section 3.2.3 (1))

Checking the program using the LIST command shows that it is actually "OK".

When a line number is changed, the old number remains as well. So, in this sense, this correction method can also be used as a copy operation.

POINT

Pressing the [Enter] key enables a program to be changed. Try different screen editing methods. Appendix 4.2 gives details about other keys used to edit a program.

3.3 Storing and Reading a Program

All programs located in the memory of the AD51H are cleared when (a) power to the AD51H is turned OFF, or (b) the NEW command is executed to create a program. Therefore, a created program must be stored in a memory card, FD, or HD (see "REMARK" on the next page).

3.3.1 Memory cards that can be used for the AD51H-BASIC

Memory cards used for the AD51H-BASIC must be formatted by giving the CFORMAT command in the AD51H system mode. The CFORMAT command is used to divide the area in the memory card into two parts:

Execution area ..... Stores (a) the programs required to automatically start up the AD51H, and (b) programs executed by the PC CPU and by the BASIC program being executed.

File storage area ..... Stores (a) data files used in a program, and (b) programs stored by using the SAVE command.

Those areas can only be accessed in the following cases:

Execution area

Programs can be registered in the AD51H system mode.

S> ■ .....Can only be used in the system mode.

File storage area

Data and programs can be registered using a BASIC command.

OK
■ ..... Can be used when a BASIC command

POINT

In this manual, all memory card input/output commands are executed to the file storage area.

REMARKS

(1) The AD51H/AD51H-S3 User's Manual (Programming and Debugging) gives details about the CFORMAT command and the AD51H system mode.

(2) Section 6. gives data file details.

3.3.2 Storing a program

Use a SAVE command to store a program to the file storage area in the memory card, FD, or HD. Use the SAVE command as follows:

SAVE Δ "[Drive Number] : [System Name \ ] [File Name]"

(1) Use the [Drive Number] to specify the memory card or FD where the program will be stored.

Memory card in MEMORY CARD [1] 0

Memory card in MEMORY CARD [2] 1

PC/AT FDD 2

PC/AT HDD [C] 3

REMARK

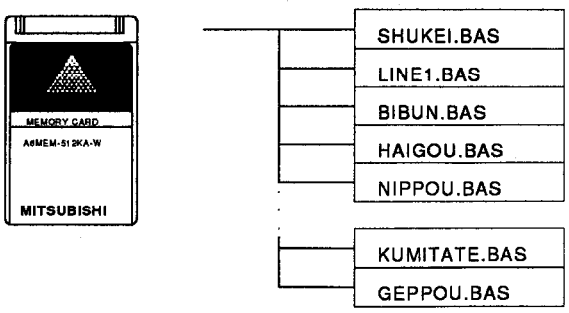
FD ..... Floppy disk is abbreviated to FD.

FDD ..... Floppy disk drive is abbreviated to FDD.
An FDD is used to access an FD.

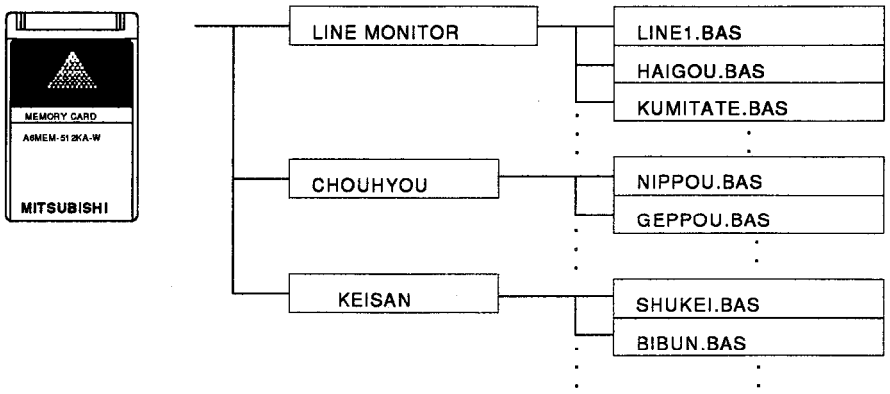
HD ..... Hard disk is abbreviated to HD.

HDD ..... Hard disk drive is abbreviated to HDD.
An HDD is built in the PC/AT.

(2) Use the [System Name] to categorize programs into groups to be managed. If a memory card or FD contains a lot of programs it is difficult to manage those programs.



Thus, when programs are grouped, it is much easier to manage them.



The system name \ ] is used for specifying group names. When a specified [system name \ ] does not already exist, that name is registered and programs are stored in that group. If the specified name already exists, programs are stored in that group.
The system name can be ignored. If this is done, programs are stored without a group name.

POINTS

(1) A system name must end with "\". If a system name does not end with "\", that system name cannot be recognized.

(2) A system name cannot contain another system name.

To specify the [system name \ ], comply with the following rule:

- Among the characters that can be used are the following symbols (as well as alphabetical and numerical characters).
  !,#,$,%,&,',(,),-,@,^,_,{,},~
- Up to eight characters can be used to specify a system name.
- The following character strings cannot be used as a system name:
  If one of them is used for a system name, the programs in that group cannot operate normally. However, these character strings can be included in a system name:
  AUX, CLOCK, CON, NUL, PRN

(3) Use the [File Name] for naming a stored program. A file name should be easily identifiable.
To specify the file name, comply with the following rules:

- Among the characters that can be used are the following symbols (as well as alphabetical and numerical characters):

  !, #, $, %, &, ', (,), -, @, ^,_, {,},~

- Up to eight characters can be used to specify a file name.
  Up to three characters can be used to specify an extension.

- The following character strings can neither be used as a file name nor as an extension If they are, the corresponding program cannot operate normally. However, these character strings can be included in a file name.

  AUX, CLOCK, CON, NUL, PRN, BAT, COM, EXE

Specification example

SAVE"0:51H\TEST1" ..... Specifies "51H" as the system name and "TEST1.BAS as the file name, and stores the program in the file storage area of a memory card (installed in MEMORY CARD1).

SAVE"2:KEISAN.BAS" .. Specifies "KEISAN.No1" as the file name without a system name, and stores the program in the FD in the PC/AT.

POINT

**Make sure that necessary programs are kept intact.
Also, programs stored in the memory cards and FDs should regularly be backed up. The SW[ ]IX-AD51HPE software package operating manual gives details about backing up data.**

3.3.3 Reading a program

Give the LOAD command to read programs stored in the file storage area of a memory card, floppy disk, or hard disk using the AD51H memory.
The LOAD command can be used as follows:

LOAD Δ "[Drive Name]: [System Name] [File Name]"

(1) [Drive Name], [System Name\], and [File Name] can be specified in the same way as the SAVE command.
See Section 3.3.1(1) to (3).

(2) If a specified [System Name\] or a specified [File Name] does not exist, that will cause a "File not found" error.

Specification example

LOAD"1:NAGATA\SUMPLE" ..... Reads the SUMPLE.BAS program from the NAGATA file in a memory card (installed in MEMORY CARD2).

LOAD"2:LINE1.P01" ..... Reads the LINE1.P01 program from the floppy disk in the PC/AT (without a system name).

POINT

Use the FILES command to check what programs exist in an floppy disk or memory card.
Section 3.4.1 gives details.

3.4 Arranging Files in the Memory Card and Floppy disk (File Order)

Data as well as BASIC programs can be stored in a memory card, floppy disk, and hard disk. (Section 1. gives details about how to store data.). Programs and data stored in a memory card and floppy disk are generically called "files". This section describes how to display a file list and how to change a file name so as to arrange files.

3.4.3 Displaying file names

A file list can be displayed to see what files exist in the file storage area of a memory card, floppy disk, or hard disk. When a file list is displayed, give the FILES command as shown below:

FILES Δ "[Drive Number]:[System Name\][File Name]"[,S]

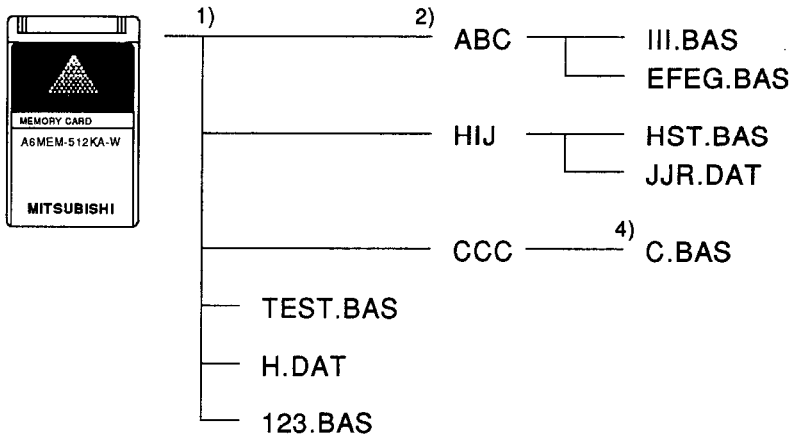
(1) Section 3.3.1(1) to (3) gives details about set contents and how to specify [Drive Number], [System Name\], and [File Name].

(2) The [System Name] and [File Name] can be omitted. A wild card can be specified as the [File Name] (ambiguous specification). Appendix 1. tells how to use wild card.

(3) When specifying the [,S] on the same line, the file size and creation date are displayed as well as the file names.

Display example

Assume that files are stored in the file storage area of a memory card (installed in MEMORY CARD2) as shown below:



1)
```
FILES"1:"
\ABC      :\HIJ    :\CCC      :TEST BAS:
H  DAT    :123 BAS:
OK
■
```
.....Displays file names which have no system name in the file storage area of the memory card.

[ ]\ indicates a system name. Therefore, the ABC, HIJ, and CCC system names are registered in this memory card.

2) `FILES"1:ABC\"` ..... Displays file names (in the ABC system name group) stored in the file storage area of a memory card.

```
III BAS:EFG BAS:
OK
■
```

3) `FILES"1:HIJ\JR.DAT"` .....Displays the JR.DAT file (in the HIJ system name group) stored in the file storage area of a memory card.

```
File not found
OK
■
```

Since the JR.DAT file does not exist in the HIJ system name group, a "File not found" error occurs.

4) `FILES"1:CCC\C.BAS", S` .....Displays detailed information about the C.BAS file (in the CCC system name group) stored in the file storage area of a memory card.

```
C   BAS   XXXX   XX-XX-XX   XX:XX
OK
■
```

- XXXX: Indicates the file size in units of bytes.
- XX-XX-XX: Indicates the date the file was created (starting with the year, followed by the month and day).
- XX:XX: Indicates the time the file was created.

3.4.2 Changing a file name

Give the NAME command to change a file name (of a program or data stored in the file storage area of a memory card, floppy disk, or hard disk) when necessary. Use the NAME command as shown below:

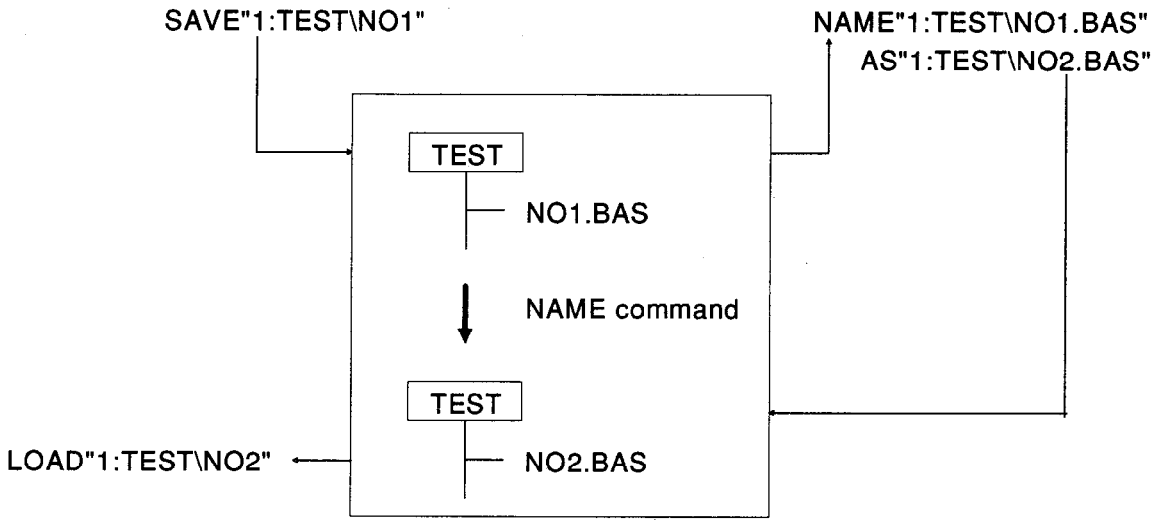
NAME Δ "[Drive Number]:[System Name\][Present File Name]"AS
"[Drive Number]:[System Name\][New File Name]

(1) Section 3.3.1 (1) and (2) gives details about the set contents and how to specify [Drive Name] and [System Name\].

(2) Specify the [Present File Name] and [New File Name] in the same manner as the [File Name] in Section 3.3.2 (3).

Change example

Assume that the name (of a file stored in the file storage area of the memory card (installed in MEMORY CARD2)) is changed from "No1.BAS" to "No2.BAS".



3.4.3 Deleting a file

The number of files that can be stored in a memory card or a FD is limited. To store more files than the capacity of a memory card or FD can handle, another memory card or FD is necessary. However, deleting unnecessary files will provide space in a full memory card or full FD for more programs or data files.

Give the KILL command to delete a file in a memory card or FD. Use the KILL command as shown below:

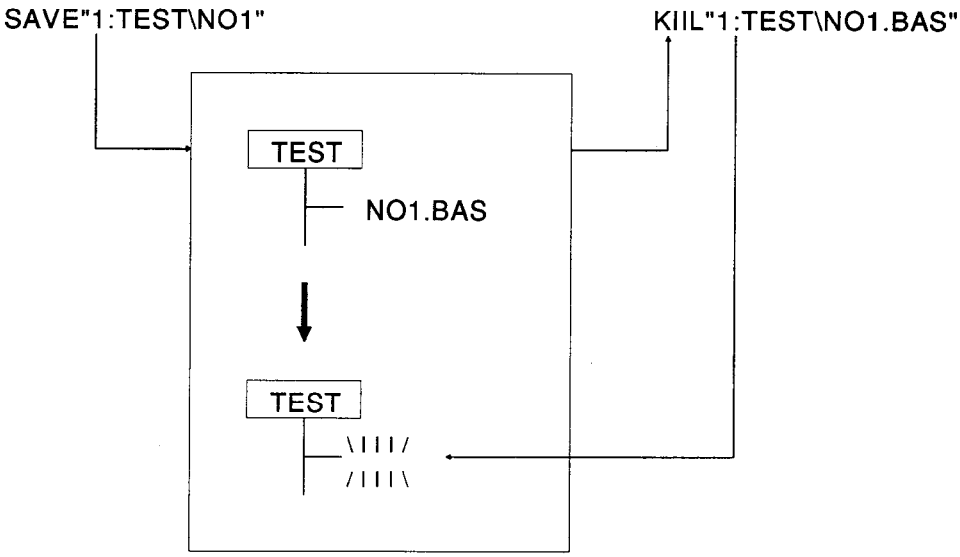
KILL Δ "[Drive Name]:[System Name\][File Name]"

(1) Section 3.3.2(1) and (2) gives details about set contents and how to specify [Drive Name] and [System Name\].

(2) Specify the [File Name] in the same manner as the [File Name] in Section 3.3.2(3).

Deletion example

Assume that the NO1.BAS (stored in the file storage area of a memory card (installed in MEMORY CARD2)) is deleted.



POINT

Once a file is deleted by giving the KILL command, that deletion cannot be undone.

REMARK

An unnecessary system name can be deleted by giving the KILL command. For details, see the KILL command explanation in this manual.

3.5 Data Setting

There are three ways to assign a value to a BASIC variable:

1) Use an assignment sentence (A=1, B=120, etc.)
2) Use a device such as keyboard to input a value.
3) Use commands READ to DATA.

Since Sections 3.11, 4, 6 and 7 give details about item 2), this section only describes items 1)and 3).

3.5.1 Assignment sentence

The "=" symbol is often used in programming. But, this is not used in the mathematical meaning (i.e., an equal sign).
First of all, run the following program:

```
10 A=1
20 PRINT A
30 A=A+2
40 PRINT A
50 END
RUN
 1
 3
OK
■
```

Look at line 30. A=A+2 can be found there. If this is regarded as a mathematical equation, the result of 0=2 can be obtained. Mathematically, this equation contradicts itself. However, in this program, it can work properly In a BASIC program, the "=" symbol is used for assigning the result on the right side of an expression to a given variable on the left side. Line 30 signifies that 2 is added to A.

Therefore, in a BASIC program, A=C+E is a correct specification, but C+E=A causes an error.

By running the two following programs, the meaning of "=" in a BASIC program will be understood:

```
10 A=5:B=7
20 PRINT A,B
30 A=B
40 PRINT A,B
50 END
RUN
 5    7
 7    7 ····· Assigns the value of B to A.
OK
■
```

```
10 A=5:B=7
20 PRINT A,B
30 B=A
40 PRINT A,B
50 END
RUN
 5    7
 5    5 ····· Assigns the value of A to B.
OK
■
```

3.5.2 Generating data in batch

Use the READ and DATA commands to generate data in batch. The DATA command is used for giving values, and the READ command is used for collecting those values. Run the following program:

```
10 READ A,B,C,D
20 PRINT A,B,C,D
30 DATA 35, 6,-8,12
RUN
35  6  -8   12
OK
■
```

As shown below, each value of the DATA command corresponds to the values of the READ command.

A	B	C	D
↑	↑	↑	↑
35	6	-8	12

The DATA command is also used for giving a single value. So, in a BASIC program, the DATA command can be used for the number of times values are given.
In this case, each variable of the DATA command corresponds to these values in ascending order.
A character string can be used as well.

```
10 DATA 51H
20 READ A$,B$,C$
30 DATA MEE
 .
 .
 .
70 DATA TD
80 READ D$
90 DATA PC
 .
 .
```

A$	B$	C$	D$
↑	↑	↑	↑
"51H"	"MEE"	"TD"	"PC"

However, the following program example is incorrect:

```
10 DATA MTC          ?
20 READ BM$,CM$
```

BM$	CM$
↑	↑
"MTC"	\ III / / III \

If a READ variable exists that does not correspond to a DATA command value, that will cause an error.

When values are specified using the DATA command, these values correspond to variables specified using the READ command as shown below:

```
10 READ    A$,B%
20 DATA    7,7
```

A$ ↕ "7"

B% ↕ 7

When the variable corresponding to "7" is specified as a numerical variable using the READ command, "7" is handled as a numerical value.

When the variable corresponding to "7" is specified as a character variable using the READ command, "7" is handled as a character string.

To utilize DATA values that have already been used, give the RESTORE command. The following gives an example of using the RESTORE command:

```
10 DATA    7,8
20 READ    A,B
30 RESTORE
40 READ C
 .
 .
 .
 .
```

A	B	C
↑	↑	↑
7	8	7

Reuse the DATA values (7, 8, and 7) on line 10 by giving the RESTORE command.

The RESTORE command can be accompanied by a line number. When a line number is specified, the operation is as shown below:

```
10 DATA    7
20 DATA    8
30 READ    A,B
40 RESTORE 20
50 READ C
```

A	B	C
↑	↑	↑
7	8	8

Reuse the DATA values (7, 8, and 8) on line 20 by giving the RESTORE command.

REMARK

When the READ command is given, an array can be used as well as a value. The following gives an example when assigning values to arrays A$(0), A$(1), and A$(2):

```
10 DIM      A$(2)
20 READ     A$(0),A$(1),A$(2)
30 PRINT    A$(0),A$(1),A$(2)
40 DATA   51H,PC,A
RUN
51H     PC    A
OK
■
```

A$(0)	A$(1)	A$(2)
↑	↑	↑
"51H"	"PC"	"A"

3.6 Jump and Repeat

A BASIC program flows according to the line number sequence. To change the execution sequence (when necessary), AD51H-BASIC contains the following commands:

GOTO ..... Represents an unconditional jump

ON GOTO ..... Represents a conditional jump

FOR to NEXT ..... Repeats a series of commands for a specified number of times.

WHILE to WEND ..... Repeats a series of commands while a condition is satisfied.

3.6.1 Unconditional jump

Use the GOTO command when directing the computer to go to a specified step.

```
10 A=1
20 PRINT A
30 A=A+1
40 GOTO 20
RUN
1
2
3
.
.
.
```

Cannot stop?

Directs the computer to go to a specified step (specified along with the GOTO command).

POINT

When the execution of a program is forcibly canceled, press the [Break] or [Ctrl] + [C] keys. The "Break in XXX" message appears, and the computer returns to the OK state.

3.6.2 Conditional jump

Use the ON-GOTO command when one or more jump commands must be specified.

```
10 INPUT "A=";A
20 ON A GOTO 40,50,60
30 END
40 PRINT "ONE" : GOTO 10
50 PRINT "TWO" : GOTO 10
60 PRINT "THREE" : GOTO 10
RUN
A=?2 [Enter]
TWO
A=?3 [Enter]
THREE
A=?4
OK
■
```

Represents that if A is "1", go to line 40.

Represents that if A is "2", go to line 50.

Represents that if A is "3", go to line 60.

Represents that if A is "4", go to the next line. The next lines is END, so the program will terminate.

POINT

The jump line varies according to the variable specified following the "ON". As many line numbers as necessary can be specified following a GOTO command.

3.6.3 Repeating a series of commands for a specified number of times

Use a FOR-NEXT sentence to execute a series of commands for a specified number of times. Look at the following program.

```
10 FOR N=1 TO 5
20 PRINT N;
30 NEXT N
40 END      (Executed five times)
RUN
   1  2  3  4  5
OK
■
```

POINT

In a FOR-NEXT sentence, a series of commands are executed until a variable reaches a specified value.

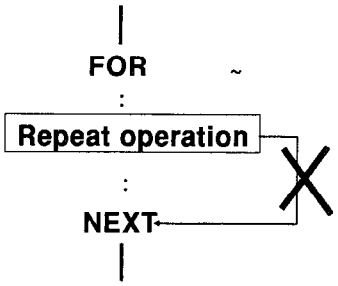
"N" can be increased not only by one, but also by a number other than one (for example, 0.5 and 2). "N" can also be decreased in the same way. Use the STEP to specify an addend or a subtrahend.

```
10   FOR N=2 TO 4 STEP 0.5
20   PRINT N;
30 NEXT N
RUN
 2  2.5  3  3.5  4
 →   →   →   →   →
OK  0.5  0.5  0.5  0.5
■
```

```
10   FOR N=5 TO 1 STEP -2
20   PRINT N;
30 NEXT N
RUN
 5  3  1
 →  →
OK -2 -2
■
```

POINT

Do not include a GOTO or IF-GOTO command in a FOR-NEXT sentence.
If this is done, the computer may not have enough memory to manage the FOR-NEXT command, and an "Out of memory" error will occur.
In the WHILE-WEND sentence shown in the next section, this does not have to be taken into consideration.



REMARK

When a FOR-NEXT sentence exists, the NEXT command determines whether or not that series of commands is repeated.
So, the FOR-NEXT sentence shown below is executed only once.

```
10 FOR J=10 TO 1 STEP 2
20 PRINT "A"
30 NEXT J
RUN
A
OK
■
```

The command on line 10 is skipped, but the command on line 20 is executed.

3.6.4 Repeating a series of commands while a condition is satisfied

Use a WHILE-WEND sentence to repeat a series of commands while a condition is satisfied.

This indicates the condition that N is less than 4.

```
10 N=0
20 WHILE N<4
30 PRINT N;
40 N=N+1
50 WEND
60 END
```

Repeats these steps.

```
RUN
 0  1  2  3
OK
■
```

These steps are repeated while the condition placed following the WHILE is specified.
Section 3.7 gives details about how to specify a conditional sentence.

The WHILE-WEND sentence is similar to the FOR-NEXT sentence, but the WHILE command determines whether or not that series of commands is repeated. So, without that condition being satisfied, the series of commands are not executed.

Loop operation using a FOR-NEXT sentence

```
10 FOR N=1 TO 1
20 RPINT N
30 NEXT N
40 END
RUN
  1
OK
■
```

Loop operation using a WHILE-WEND sentence

```
10 N=1
20 WHILE N<1
30 PRINT N
40 WEND
50 END
RUN
OK
■
```

The next command determines the execution of loop.

3.7 Using a BASIC Conditional Sentence

In a BASIC program, a conditional sentence can be used to cause a branch.

3.7.1 Specifying conditions

Use the following symbols to specify conditions:

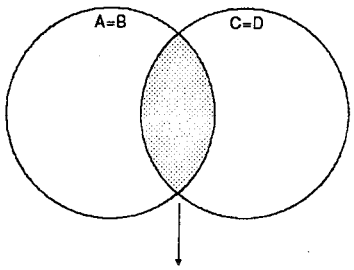
Symbol	Meaning	Example	
=	Equal	A=B	Satisfied when A is equal to B.
<	Less	A<B	Satisfied when A is less than B.
>	Greater	A>B	Satisfied when A is greater than B.
<= =<	Equal or less	A<=B ...	Satisfied when A is equal to B or A is less than B.
>= =>	Equal or greater	A>=B ...	Satisfied when A is equal to B or when A is greater than B.
<>	Not equal	A<>B ...	Satisfied when A is not equal to B.

The AND and OR symbols can be used to make various conditional sentences. Such a sentence which contains conditions (combined using the AND or OR symbol) is called a logical expression.

- Satisfied when A is B and C is D: Combine those conditions using the logical symbol (AND). As many conditions as necessary can be combined in an expression.

  Example

  A=B AND C=D ..... Satisfied when A is equal to B and C is equal to D.

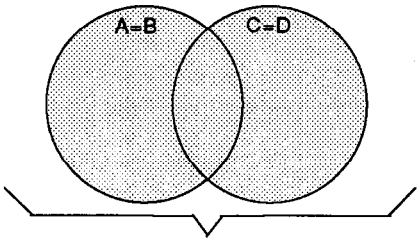


The range in which the logical expression is satisfied.

- Satisfied when A is B or C is D: Combine those conditions using the logical symbol (OR). As many conditions as necessary can be combined in an expression.

Example

A=B OR C=D ..... Satisfied when A is equal to B or C is equal to D.



The range in which the logical expression is satisfied.

3.7.2 Branch command

Use the IF command to cause a branch as shown below:

IF	THEN	ELSE
Condition; or Logical expression	Instruction executed when the condition or the logical expression is satis-	Instruction executed when the condition or the logical expression is not satisfied.

(1) The [Condition or Logical expression] contains a condition or a logical expression as shown in Section 3.7.1.

(2) Following the THEN, place a command executed when the condition or logical expression is satisfied. A multistatement (see section 2.3) can be used instead of a command.

(3) Following the ELSE, place an command executed when the condition or logical expression is not satisfied. A multistatement (see section 2.3) can be used instead of the command.

Example

Line 20 does not have to be divided into two parts. This can be input on a single line.

```
10   FOR  I=1 TO 10
20   IF  I=<5 THEN PRINT I ; "Equal or less than"
                                                    ..... Executed when I is equal
                                                          to 5 or less than 5.
             ELSE PRINT I ; "Greater than"
                                                    ..... Executed when I is greater
                                                          than 5.
30   NEXT I
40   END
RUN
 1   Equal or less than
 2   Equal or less than
 .
 .
 5   Equal or less than
 6   Greater than
 .
 .
10   Greater than
OK
■
```

REMARKS

(1) When the GOTO command is placed following the THEN or ELSE, the GOTO command can be omitted as shown below:

Example

IF A=1 THEN GOTO 100 ELSE <u>GOTO</u> 200

⇩

IF A=1 THEN 100 ELSE 200

IF A=1 THEN GOTO 70 ELSE GOTO 10

⇩

IF A=1 GOTO 70 <u>ELSE</u> 10

↑

ELSE cannot be omitted.

(2) If a single-precision value is compared with a double-precision value, the result can be wrong.

Example

```
10   A=0
20   FOR I=1 TO 1000
30   A=A+0.001                  ..... Since 0.01 is added to A 1000
40   NEXT I                            times, A should be 0.1.
50   IFA=0.1 THEN PRINT "EQ"    ..... If A is 0.1, then "EQ!" will be dis-
60   END                               played.
RUN
OK   ..................................... Does not display "EQ!".
PRINT A
0.100001   ............................... A is not 0.1.
OK
■
```

This is because single-precision and double-precision values are stored in the floating-point format. Some values cannot be retained correctly (as is true of the above-mentioned).

To see if those values are equal, compare them using the absolute difference of those values.

```
10   A=0
20   FOR I=1 TO 10
30   A=A+0.01
40   NEXT I
50   IF ABS(0.1-A)<1E-5 THEN PRINT "EQ"   ..... If the difference between 0.1 and
60   END                                         A is less than 10^-5, those values
RUN                                              are considered to be equal.
EQ!  ..................................... Displays "EQ!".
OK
■
```

3.8 Using an Array

Assume that 100 values are assigned to variables. For example, use variables A1, A2,....

```
10 A1=52
20 A2=60
30 A3=17
    :
    :
    :
```

In this case, 100 variables are necessary. Since it is hard to handle them, it is convenient to use an array. Before using an array, the BASIC should be notified that the array is used. Use the DIM command to do so.

```
DIM A(99)
```

Giving a DIM command provides a set of array elements as shown below:

A(0)	A(1)	A(2)	A(3)	.....	A(98)	A(99)

If "70" is assigned to A(50), the expression will be A(50)=70.

```
A(50)=70
```

The value in parentheses can be replaced by a variable.

```
S=50:A(S)=70
```

If the expression is S=50:A(S)=70, "70" will be assigned to A(50) since the S in parentheses is 50.
Now, retry assigning 100 values to variables. When the READ, DATA, and FOR-NEXT shown in Sections 3.5 and 3.6 are used:

```
10 DIM A (99)
20 FOR I=0 TO 99
30 READ A(I)
40 NEXT I
50 DATA 52,60,70, ...
```

The program can be much shorter than without those commands. Programming can be made easier by assigning a variable to a subscript.

An array can be used to handle alphabetical characters as well as numerical ones. When an array is used for handling numerical characters, select the data type among integer, single precision, and double precision.
(To define the variable type, see the DEF, INT/DEF, SNG/DEF, and DBL commands.)

A$(n) ..... Represents character array A$.

A%(n) ..... Represents integer array A%.

A!(n) ..... Represents single-precision array A!.

A#(n) ..... Represents double-precision array A#.

3.8.1 Number of array dimensions

The number of dimensions refers to the number of subscripts in parentheses. For example, a three-dimensional array is shown below:

One-dimensional
..... A$(x)

A$(0)
A$(1)
A$(2)

Two-dimensional
..... B(x,y)

B(0,0)	B(1,0)	B(2,0)
B(0,1)	B(1,1)	B(2,1)
B(0,2)	B(1,2)	B(2,2)

Three-dimensional
..... C(x, y, z)

C(0,0,0)	C(1,0,0)	C(2,0,0)
C(0,1,0)	C(1,1,0)	C(2,1,0)
C(0,2,0)	C(1,2,0)	C(2,2,0)
C(0,2,1)	C(1,2,1)	C(2,2,1)
C(0,2,2)	C(1,2,2)	C(2,2,2)

By using an array with two or more dimensions, different records of data can easily be referred to. For example;
Use array A(x, y) in the following specifications:

x=1..... PC CPU

x=2 ..... FA controller

x=3 ..... Robot

y=1 ..... Planned production volume

y=2 ..... Actual production volume

y=3 ..... Percentage of actual volume/planned volume

Actual praduction volume of PC CPU **A(1, 2)**

Planned production volume of Robot **A(3, 1)**

By using this array, it is easy to refer to a necessary record.

There can be up to 255 array dimensions.

REMARK

If more than the maximum number of array dimensions (255) are specified for an array, it can cause a memory shortage.

3.9 Using a Subroutine

In a program, the same process can be required many times. For example, assume that bar graphs are drawn on the screen using "*".

```
 10   READ A
 20   FOR I=1 TO A
 30   PRITN "*";
 40   NEXT I
 50   PRINT
 60   READ A
 70   FOR I=1 TO A
 80   PRINT "*";
 90   NEXT I
100   PRINT
110   DATA 5,9
120   END
```

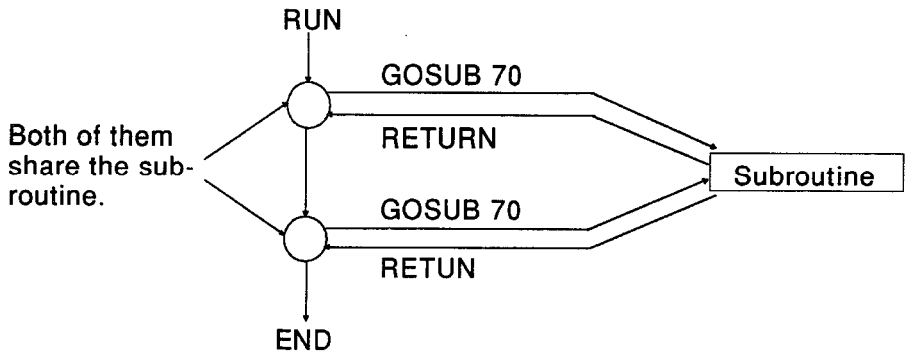
```
RUN
*****
*********
OK
■
```

In this program, the parts contained by ☐ are the same. If this program contains more commands, it will be extremely long.
The subroutine is a group of commands that perform a specific processing function. The subroutine can be used by giving the GOSUB and RETURN commands. Change the above program as shown below:

```
 10   READ A
 20   GOSUB 70
 30   READ A
 40   GOSUB 70
 50   DATA 5,9
 60   END
 70   FOR I=1 TO A
 80   PRINT "*";
 90   NEXT I
100   PRINT
110   RETURN
```

```
RUN
*****
*********
OK
```

The same results are obtained, but one of the same parts can be deleted. The GOSUB 70 commands on lines 20 and 40 call the subroutine (contained by the ☐ beginning on line 70. After finishing the subroutine process, the RETURN command returns the computer to line 110.



REMARK

To call a subroutine, be sure to use the GOSUB command. If the GOTO command is used, an error will occur when the computer returns to the main routine.

3.10 Displaying a Character on the Screen

Use the PRINT command to display a character on the screen. Since the print is the most frequently used command in BASIC, "?" can also be used instead of the PRINT command in a program.

Example

```
OK
PRINT "ABC"
ABC
OK
```

⇨

```
OK
?"ABC"
ABC
OK
```

Placing contents after the PRINT command.
Contents that can be displayed are constants, variables, arrays, functions, and their combinations using operators.

Example

PRINT "ABC" ..... Displays the constant of "ABC".

PRINT A% ..... Displays the content of the A% integer.

PRINT A$+C$ ..... Displays the result of the addition of character variables A$ and C$.

Place commas or semicolons between values to display several contents on a single line. The following example shows semicolons being used.

Example

```
OK
A=100
OK
B=-50
OK
PRINT A;B;A+B
Δ100 Δ-50 ΔΔ50 Δ
```

Content of A
Content of B
Content of A + B

```
OK
A$="ABCD"
OK
B$="EFGH"
OK
PRINT A$;B$;A$+B$
ABCD EFGH ABCDEFGH
```

Content of A$
Content of B$
Content of A$ + B$

When semicolons are placed between variables as shown below, the contents can be continuously displayed. However, numerical values and characters are displayed in different formats.
Numerical values are displayed as shown below:

- Numerical values are always displayed along with a sign:

  When a value is negative: Accompanied by "-"

  When a value is positive: Accompanied by "Δ" (A space is given before the value)

- When a value is displayed; a space is automatically given after the value (By taking these principles into consideration, the above example is easy to understand.)

  Therefore, placing semicolons between values results in automatically putting spaces between values on the screen.

  When those values are characters, they are continuously displayed on the screen.

  When commas are placed between variables, their contents are displayed as shown below:

  Example

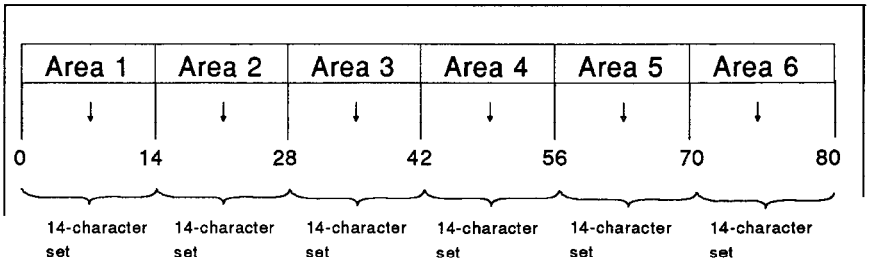
```
OK
A=100
OK
B=-50
OK
PRINT A,B,A+B
Δ100 Δ   -50 Δ  Δ 50 Δ
```

14-character set 14-character set

```
OK
A$="ABCD"
OK
B$="EFGH"
OK
PRINT A$,B$,A$+B$
ABCD     EFGH      ABCDEFGH
```

14-character set 14-character set

To control a screen in BASIC, a screen line is divided into six 14-character sets.



Placing commas between values displays those characters in units of character sets on a screen line. When a numerical value is displayed, the display format is the same with the semicolon.

3.10.1 Functions to display a character

The PRINT command has two dedicated functions.

(1) TAB function

Use the TAB function to display values so that the display location on a screen line can be specified for each value. Place semicolons to use several functions on a single line.

Example

```
PRINT "A";TAB (12);"B";TAB(27);"c"
A           B           C
```

12 Characters

27 Characters

```
10 INPUT A$
20 INPUT B$
30 PRINT A$,B$
40 PRINT A$;TAB(20);B$
RUN
? 1234567
? 3478
1234567 3478
1234567                 3478
OK
RUN
? 123456789012345
? 1000
123456789012345                 1000
123456789012345         1000
OK
■
```

Those can be aligned by using a TAB function.

When a TAB function (unlike a comma or semicolon) is used, the display location does not vary according to the display contents.

(2) SPC function

Use the SPC function to give a specified number of spaces following a previously-specified value on the screen. To combine another SPC function, place a semicolon after the previous one.

When an SPC function (unlike a comma or semicolon) is used, the display location does not vary according to the display contents.

Example

```
PRIMT "A";SPC(6);"B";SPC(5);"C"
A      B       C
```

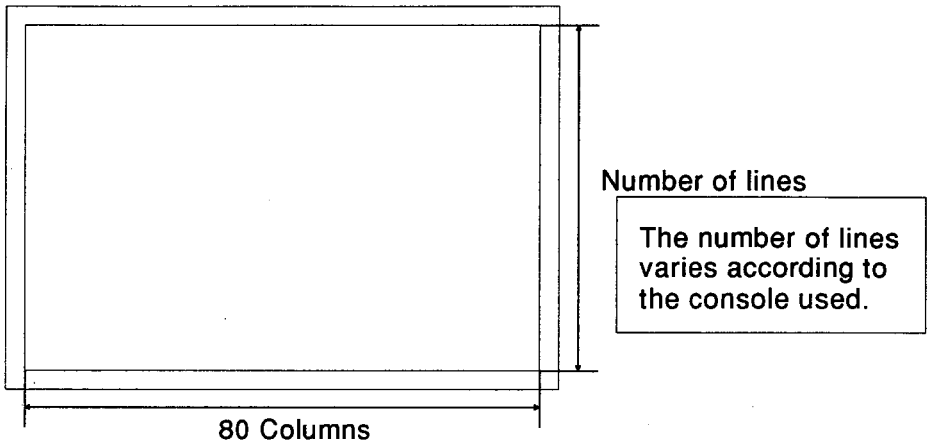
6 Characters 5 Characters

```
10 INPUT A$
20 INPUT B$
30 PRINT A$,B$
40 PRINT A$;SPC(5);B$
RUN
? 1234567
? 3478
1234567          3478
1234567      3478
OK
RUN
? 123456789012345
? 1000
123456789012345      1000
123456789012345      1000
OK
```

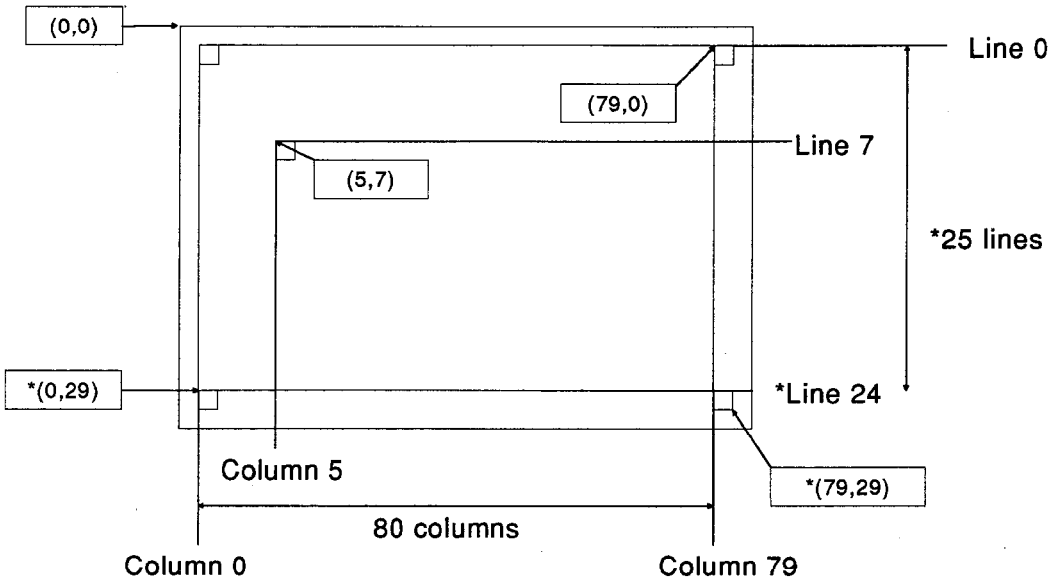
The same number of spaces are inserted by using a SFC function.

3.10.2 Displaying a character at a given position

Horizontally, the display location can be changed by using a comma, semi-colon, or TAB/SPC function. To change the display location vertically, use the LOCATE command. The display specifications are shown below:



When the LOCATE command is used, the display location can directly be specified since the screen is regarded as X-Y coordinates (the left top on the screen corresponds to coordinates (0, 0)). The X coordinate refers to the horizontal position. The Y coordinate refers to the vertical position.



* This example assumes that a PC/AT is used as the console. If another machine is used as the console, the vertical (Y) coordinate can vary.

The following describes how to use the LOCATE command.

LOCATE Δ Horizontal position, Vertical position

By giving this expression, the display location can be specified. Thereafter, the PRINT command should be used.

Example

```
10 CLS ..................... Clears the screen.
20 LOCATE 10,5
30 PRINT "MELSEC AD51H-BASIC"
40 LOCATE 65,20
50 PRINT
60 END
RUN
```

```
MELSEC AD51H-BASIC



OK
■
```

3.11 Data Input from the Keyboard

There are two ways to transfer data input from the keyboard to a BASIC program Use the INPUT command to interrupt the execution of the BASIC program; or Use the INKEY$ command to check how the keyboard is working.

To interrupt the execution of a BASIC program while data is being input from the keyboard, use the INPUT command as shown below:

INPUT Δ Variable (to which a value input from the keyboard is assigned)

Example

```
10 INPUT A
20 INPUT B
30 PRINT A+B
40 END
RUN
? 6 ----→ Input the value assigned to A. For example, if "6" is as-
? 10---┐  signed to A, input "6", and press the [Enter] key.
 16    │
       └─ Input the value assigned to B. For example, if "10" is
OK        assigned to B, input "10", and press the [Enter] key.
■
```

However, when only "?" is displayed on the screen, this "?" may not be understandable. To prevent this problem, the INPUT command can be displayed along with a character constant. Consider the following program:

Example

```
10 INPUT "A=";A
20 INPUT "B=";B
30 PRINT A+B
40 END
RUN
A=?6
A=?10
 16
OK
■
```

As shown above, by placing a character constant along with a semicolon before a variable, the character is displayed as well as "?". That shows what value should be input.

When the INPUT command is used, the execution of a program is interrupted until data is input and the [Enter] key is pressed. To avoid this interruption, use the INKEY$ function. Consider the following program:

Example

```
10 A$=INKEY$
20 PRINT A$
30 GOTO 10
RUN
 :
 :
 :
```

Executing this program clears the screen. Thereafter, press the [A] key.

```
 :
 :
 :

Δ
A ←— Pressing the [A] key displays "A" on the screen.
A ←— Releasing the [A] key clears the screen.
 :
 :
 :
```

As shown above, the INKEY$ function can only be applied for verifying how the keyboards is working. Thus, the execution of a program is not interrupted. When the INKEY$ function is used, pressing a key assigns its value to a variable. This value, however, is a character string. The variable should be a character variable or a character array. Unlike the INPUT command, some characters cannot be input at once.

Example

```
10 N=0
20 PRINT "Counting"
30 N=N+1
40 K$=INKEY$
50 IF INKEY$="Y" THEN GOSUB 70
60 GOTO 30
70 PRINT "N=";N;""
80 RETURN
```

```
RUN
Counting
N= 30 ←— Press the [Y] key.
N=530 ←— Press the [Y] key.
```

POINT

If the INKEY$ function is used to suspend the input as shown below, that will lower the multitask efficiency.

```
10 PRINT "Press the [Y] key."
20 K$=INKEY$
30 IF K$="Y" GOTO 50
40 GOTO 20
50 END
```

In this case, use the INPUT function.

```
10 PRINT "Press the [Y] key."
20 K$=INPUT$(1)
30 IF K$<>"Y" GOTO 20
40 END
```

3.12 Printout (Using a Printer)

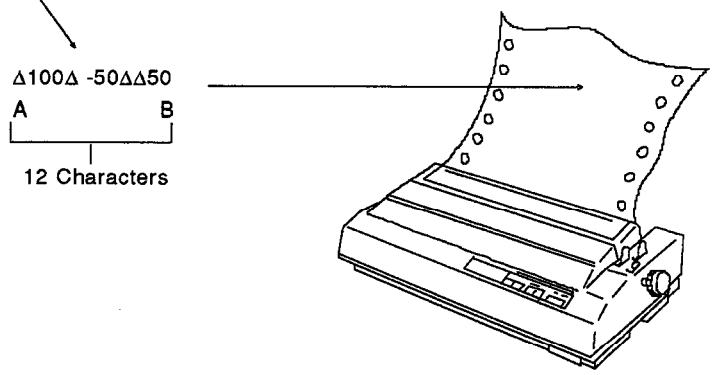
Constants, variables, and arrays can be output to a printer as well as to a console. To output data to a printer, specify the printer location by using the ZLDV command. Set the printer as shown below:

- When a printer connected to the PC/AT is used ..... ZLDV 0
- When a printer is connected to RS-232C interface channel 1 of the AD51H ..... ZLDV 1
- When a printer is connected to RS-232C interface channel 2 of the AD51H ..... ZLDV 2
- When a printer is connected to RS-422 interface channel 3 of the AD51H ..... ZLDV 3
- When a printer is connected to PARALEL interface channel 4 of the AD51H ..... ZLDV 4

The data to be printed out must be specified using the LPRINT command. The LPRINT command is used in the same manner as the PRINT command (see Section 3.10). The SPC and TAB functions can be used along with the LPRINT command. The LOCATE command, however, cannot be used with the printer at all.

Example

```
10 ZLDV 0
20 A=100:B=-50
30 LPRINT A;B;A+B
40 LPRINT "A";TAB(12);"B"
50 END
RUN
OK
■
```



REMARKS

(1) The printer can be used in different ways by sending data called "control code". For details, see the appropriate printer operating manual.

(2) By giving the ZCNTL command, it is possible to get information about the printer state and to transmit the reset signal to the printer.

3.13 Characters that are Processed in a BASIC Program

3.13.1 Character type

Characters that are processed in a BASIC program can be divided into two major groups:

- 1-bite characters ..... Can be input, consisting of alphanumeric characters and special symbols.
- Control characters ..... Consist of character codes 00_H to $1F_H$. Normally, those characters are not used. They cannot be input from the keyboard. By using the control characters, the same functions with the arrow keys [↑],[↓],[←],[→]and [Ctrl] + [ ]keys are available. To specify a control character, use the CHR$ function.

3.13.2 Processing in units of characters

(1) Taking component characters out of a character string

There are three functions for taking component characters out of a character string in order to make a new character string.
Consider the following program:

Example

```
10 A$="ABCDEFGHIJ"
20 B$=LEFT$(A$,4)
30 PRINT B$
RUN
ABCD
OK
■
```

The LEFT$ function of (A$, n) is used for taking characters (from 1 to n (viewed from the left)) out of the A$ character string and making a new character string.

Example

```
10 A$="ABCDEFGHIJ"
20 B$=RIGHT$(A$,4)
30 PRINT B$
RUN
GHIJ
OK
■
```

The RIGHT$ function of (A$, n) is used for taking characters (from 1 to n (viewed from the right)) out of the A$ character string and making a new character string.

Example

```
10 A$="ABCDEFGHIJ"
20 B$=MID$(A$,4,3)
30 PRINT B$
RUN
DEF
OK
■
```

The MID$ function of (A$, n, m) is used for taking characters (from 1 to n (viewed from the left)) out of the A$ character string and making a new character string.

•The MID$ command can be used to alter a character string. For details, see the MID$ explanation.

(2) Checking the length of a character string

Use the LEN function to check the length of a character string.

Example

```
10 A$="ABCDE"
20 A=LEN(A$)
30 PRINT A
40 END
RUN
 5
OK
■
```

(3) Encoding a character

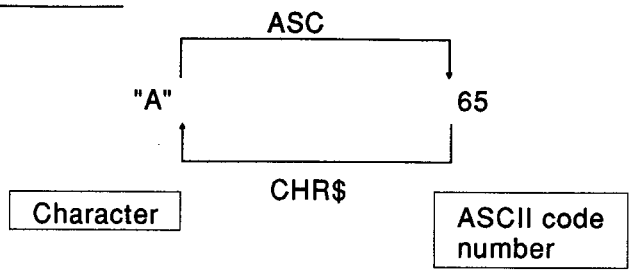
Each character used in a BASIC program corresponds to ASCII code numbers. (See Appendix 4.1.)
Alphabets, for example, correspond to code numbers as shown below:

[Character]	[Code]
A	65
B	66
⋮	⋮

Use the ASC and CHR$ functions to convert between a character and an ASCII code number.

Example

```
OK
PRINT ASC("A")
 65
OK
PRINT CHR$(65)
A
```



POINTS

(1) In the parentheses following the ASC functions, nothing but a character array or a character variable can be specified. In the parentheses following the CHR$ functions, however, nothing but a numerical constant, a numerical array, or a numerical variable can be specified.

(2) If the value in parentheses of the CHR$ function is 31 or less, it will be regarded as a control character.

(4) Converting between a numerical value and a character

If a character string consists of only numerical characters, the following calculation cannot be made.

Example

```
OK
A$="12345"
OK
B=1156
OK
PRINT A$+B
Type mismatch
OK
■
```

To handle such a character string as a numerical value, use the VAL function. The VAL function can convert a character string into a numerical value.

Example (continued from the above example)

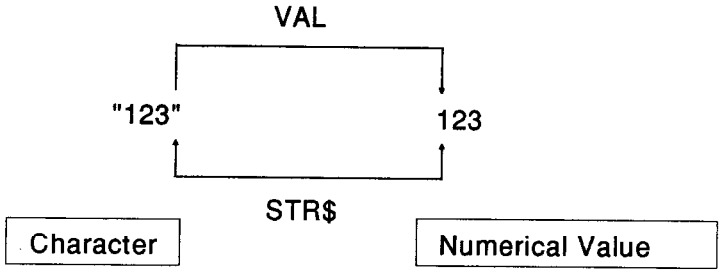
```
A=VAL(A$)
OK
PRINT A+B
 13501
OK
■
```

However, use the STR$ function to handle a numerical value as a character string. The STR$ function can be used to convert a numerical value to a character string.

Example (continued from the above example)

```
C$=STR$(A+B)
OK
PRINT RIGHT$(C$,3)
501
OK
■
```

The above descriptions can be summarized as shown below:



POINT

If a character string to be converted using the VAL function contains a character other than numerical values, +, -, E, and ·, an Illegal function call error will occur.

3.14 Numerical Functions

In a BASIC program, there are functions to find a solution to trigonometric and exponential functions. The following shows how to express those functions in a BASIC program:

POINT

To find a solution to trigonometric function, assume that the angle is expressed in radian.

Radian = Angle x π/180

(1) Trigonometric functions

- sin x ..... SIN (X)
- cos x ..... COS (X)
- tan x ..... TAN (X)
- $\tan^{-1}\alpha$ ..... ATN (A)

(2) Exponential and Logarithmic functions

- e^x ..... EXP (X)
- logeX ..... LOG (X) [natural logarithm]

There is no function that can directly find a solution to $\log_{10}X$ (common logarithm). Use a $\log_{10}X$ = logeX/loge10.

(3) Square root

- $\sqrt{X}$ ..... SQR(X)

(4) Numerical conversion

- Integer conversion ..... INT (X)
- Absolute value ..... ABS (X)
- SignSGN (X) ..... SGN(X)
- Decimal part rounding off .... FIX (X)

(5) Random number

- Random number generation using values 0 to 1 ..... RND (X)

See the specific explanations for details about the arguments of each function.

3.15 Dividing and executing very large programs

Programs larger than the memory capacity cannot be executed. However, large programs can be divided into small programs and stored. Then, reading those divided programs sequentially yields the same result as could have been obtained from the original program.
AD51H-BASIC executes a divided program as shown below:

1. When variable contents do not have to be transferred ... Use the LOAD or RUN command.
2. When variable contents have to be transferred ..... Use the CHAIN command.
3. When part of the program is shared ..... Use the CHAIN command.

(1) When variable contents do not have to be transferred

When variables that are being used by the present program do not have to be transferred to the next program, use the RUN command or the LOAD command R option to run the divided program sequentially.

```
10  PRINT "Executing Program B !!"
20  PRINT "A=";A
30  END
SAVE "PRO-B"
OK
NEW
10  PRINT"Executing Program A !!"
20  A=100
30  PRINT "A=";A
40  RUN "0:PRO-B"
SAVE "PRO-A"
OK
■
```

..... The same result is obtained when LOAD"0:PRO-B",R is executed.

```
RUN
"Executing Program A !!"
A= 100
"Executing Program B !!"
A= 0
OK
■
```

..... The variable contents have been transferred.

(2) When variable contents have to be transferred

When all variables that are being used by the present program have to be transferred to the next program, use the CHAIN command (ALL option).When some of the variables that are being used by the present program have to be transferred to the next program, use the COMMON command.

```
10  PRINT "Executing Program B !!"
20  PRINT "A=";A
30  END
SAVE"PRO-B"
OK
NEW
OK
10  PRINT "Executing Program A !!"
20  A=100
30  PRINT "A=";A
40  CHAIN"0:PRO-B",10,ALL
SAVE "PRO-A"
OK
```

..... Assumes that all variables are transferred.

```
RUN
"Executing Program B !!"
A=100
"Executing Program A !!"
A=100
OK
```

..... The variable contents have been transferred.

(3) When part of the program is shared

When part of the program being executed is replaced by part of another program, use the MERGE and DELETE options of the CHAIN command.

```
70    PRINT "Executing Program B !!"
80    A=4:GOSUB 1000
90    END
SAVE "PRO-B"
OK
NEW
OK
10    PRINT "Executing Program A !!"
20    A=7:B=3:GOSUB 1000
30    CHAIN MERGE "0:PRO-B",70,ALL,DELETE 10-30
1000  PRINT "A=";A
1010  PRINT "B=";B
1020  RETURN
SAVE "PRO-A"
OK
```

Deletes lines 90 to 30 of the program being executed, then inserts part of "PRO-B.BAS".

```
RUN
"Executing Program B !!"
A=7
B=3
"Executing Program A !!"
A=4
B=3
OK
LIST
70    PRINT "Executing Program B !!"
80    A=4:GOSUB 100
90    END
1000  PRINT  "A=";A
1010  PRINT  "B=";B
1020  RETURN
OK
```

Part of "PRO-B.BAS"

Original part of the program

POINT

When the CHAIN command is used along with the DELETE or MERGE option, a variable can be transferred by using the ALL option of the COMMON command.

4. DATA TRANSFER WITH THE SEQUENCER AND WITH BUFFER MEMORY

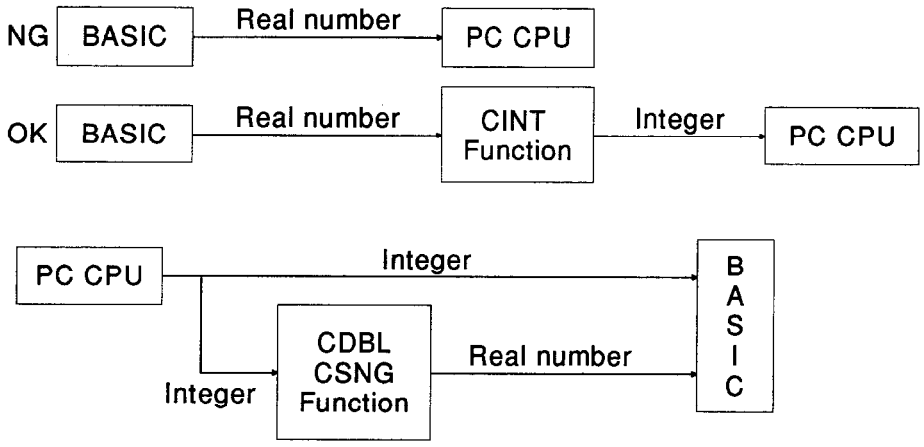
4.1 PC CPU Numerical Data and BASIC Numerical Data

The values that the PC CPU can handle are integers, ranging from -32768 to 32767 in units of one word (16-bit), and from -2147483648 to 2147483647 in units of two words (32-bit).

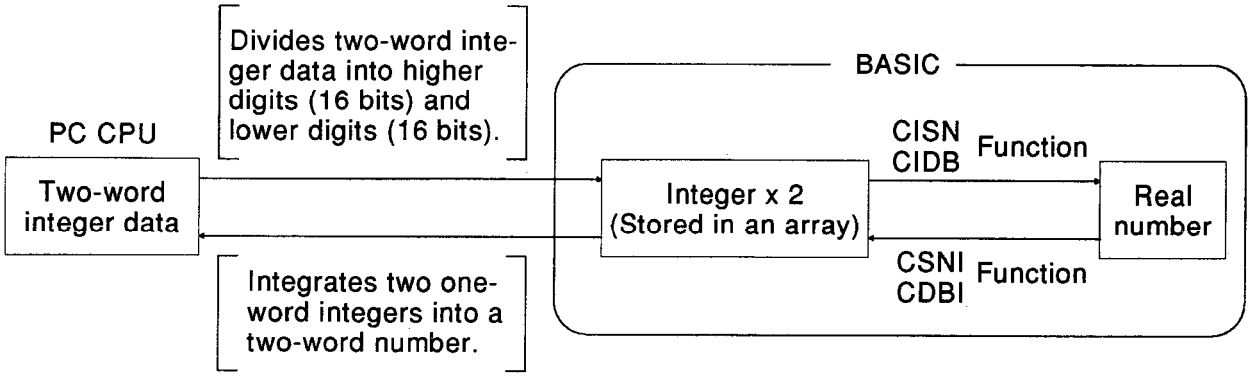
The values that BASIC can handle are real numbers ranging from 10^{-38} to 10^{38}, and integers ranging from -32768 to 32767.

As described above, since there is a difference between the way BASIC and a PC CPU handle values, conversions between real numbers and integers are necessary.

- Handling data in units of one word



- Handling data in units of two words



4.2 Data Communications between a PC CPU and BASIC

Data can be communicated between a PC CPU and the BASIC as shown below:

- Reads/writes data from/to devices in a PC CPU
- Reads/writes data from/to the buffer in a special-function module
- Reads/writes PC CPU control data
- Reads/writes sequence program and parameter data

Use the PCRD and PCWT commands to execute those operations.

POINT

The AD51H-BASIC can handle the floating-point number data used in an A2A/A3ACPU.
The explanations of the CVSMBF and MKSMBF functions in this manual give details.

4.2.1 Control table

A great many parameters must be used to execute the PCRD and PCWT commands.
To do so, use the "control table". In a BASIC program, part of the memory is used as a control table for transferring parameters (used for notifying the AD51H of the operation status).
In a BASIC program, necessary parameters are stored in an array. Specifying the array name transfers the parameters to the AD51H.
For instance, to read data stored from the devices in a PC CPU, the parameters corresponding to PC CPU station number, read device, bit/word, read head device, read number of points, and data storage variable must be specified.
If a great many parameters follow a command, it is difficult to read the program.
Thus, it is easier to read the program by using the control table.

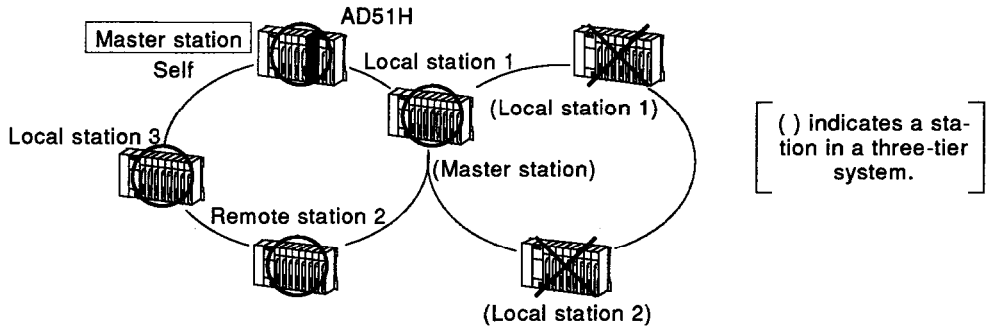
Using the controltable

```
100  RD%(0)=1            :'Station number
110  RD%(1)=1            :'Read specification
120  RD%(2)=2            :'Word unit
130  RD%(3)=1            :'Device code
140  RD%(4)=&H10         :'Head device No.
150  RD%(5)=1            :'One-word read
200  PCRD RD%( ),A%( )
```

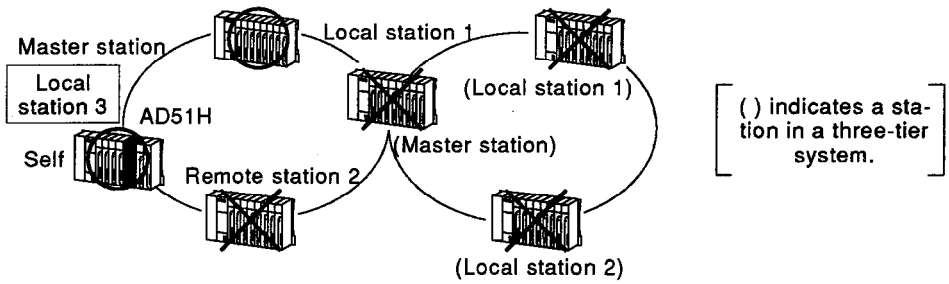
4.2.2 PC CPU station numbers

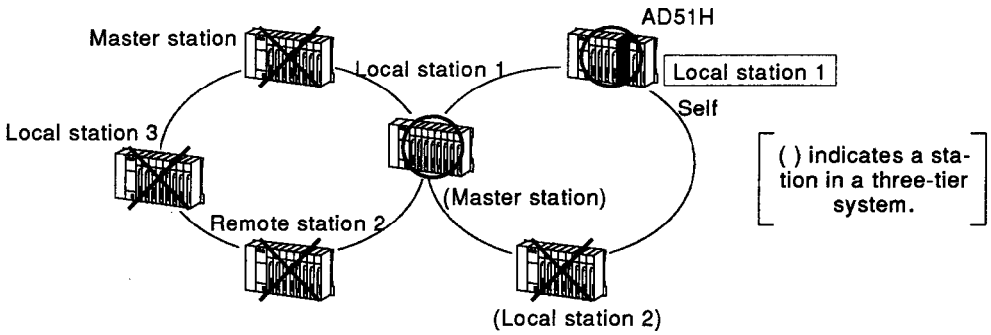
The AD51H-BASIC can not only access the PC CPU in which the AD51H is installed, but also other stations (PC CPU) linked by the MELSECNET. However, access depends on whether the PC CPU (in which the AD51H is installed) is a master or local station.

(1) If the CPU (self) in which the AD51H is installed is a two-tier master station, the AD51H-BASIC can access the self and all local and remote I/O stations in the two-tier loop.
If the CPU works as a remote I/O station, the AD51H-BASIC can access the buffer.

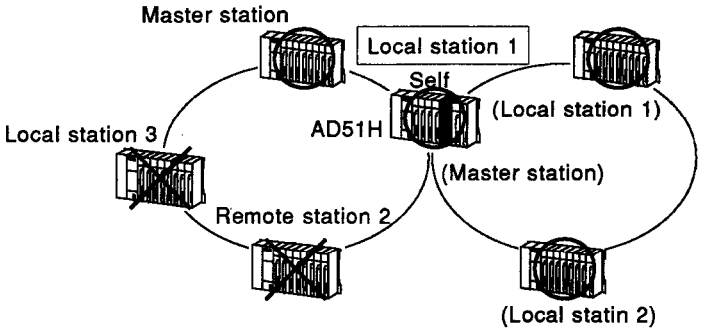


(2) If the CPU (self) in which the AD51H is installed is a two-tier or three-tier local station, AD51H-BASIC can access the self and the master station in the same loop as the self.





(3) If the CPU (self) is a three-tier master station, the AD51H-BASIC can access the self, a two-tier master station, and all local and remote I/O stations in the three-tier loop.
If the CPU is a remote I/O station, the BASIC can access only the buffer of a special-function module.



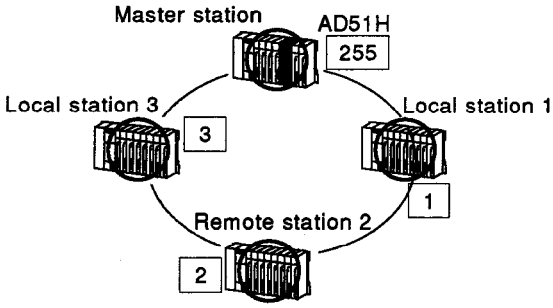
[( ) indicates a station in a three-tier system.]

To specify the PC CPU to be communicated with, set the station numbers in the same loop as the self.

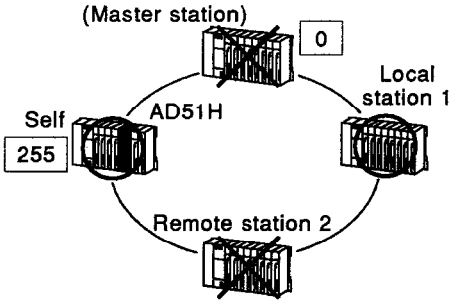
Master station (station No.0) ⟶ Set "0".
Local station 1 (station No.1) ⟶ Set "1".
Local station 2 (station No.2) ⟶ Set "2".
.
.
.
Local station 64 (station No.64) ⟶ Set "64".

When the self is accessed, set "255" as the station number.

Example 1

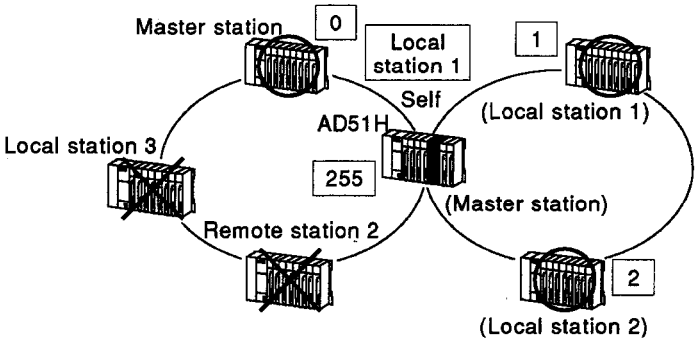


Example 2



[ ]indicates a specified station number.

Example 3

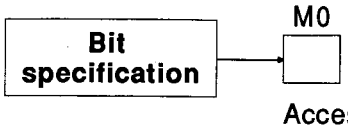


4.2.3 Selecting the processing mode

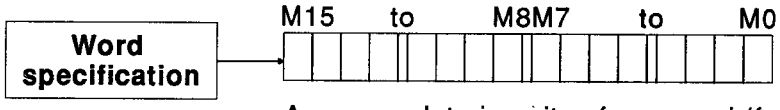
To give an operation request using the PCRD and PCWT commands, specify a value called "operation code".The explanations of the PCRD and PCWT commands in this manual give details.

4.2.4 Bit/Word specification

When a BASIC program accesses a bit device of a PC CPU, specify whether the data will be processed in units of one bit or one word (16-bit).



Accesses data in units of one bit (for one bit)



Accesses data in units of one word (for 16 bits)

If processing in units of word is specified, data will be processed in units of 16 bits.
If data is processed in units of word, specify 0 or a multiple of 16 as the head device number.

Example

If M is used: Specify M0, M16, or M32 as the head number.

If B is used: Specify B0, B10, or B20 as the head number.

POINT

To use a special-relay M to process data in units of word, specify "9000" or "9000" + a multiple of 16 as the head number.
(For instance, devices such as M9000, M9016, and M9032 can be specified.)

Whenever a word device is accessed, process data in units of word.

REMARK

Access refers to such input/output processing as data read/write.

4.2.5 Specifying a device number

Specify devices used for a PC CPU using "device code" and "device number".

The correspondence of device codes to usable device numbers is shown below:

Device Codes	Devices	Usable Device Numbers
1	X(Input)	0_H to $7FF_H$
2	Y(Output)	0_H to $7FF_H$
3	M(Internal relay)	0 to 2047 (0 to 8191)
	L(Latch relay)	0 to 2047 (0 to 8191)
	S(Step relay)	0 to 2047 (0 to 8191)
	M(Special relay)	9000 to 9255
4	B(Link relay)	0_H to $7FF_H$
5	F(Annunciator)	0 to 255
6	T(Timer [contact])	0 to 255 (0 to 2047)
7	T(Timer [coil])	0 to 255 (0 to 2047)
8	C(Counter [contact])	0 to 255 (0 to 2047)
9	C(Counter [coil])	0 to 255 (0 to 2047)
16	T(Timer [current value])	0 to 255 (0 to 2047)
17	C(Counter [current value])	0 to 255 (0 to 2047)
18	D(Data register)	0 to 1023 (0 to 6143)
	D(Special register)	9000 to 9255
19	W(Link register)	0_H to $3FF_H$ (0_H to FFF_H)
20	R(File register)	0 to 8191

( ) indicates a device number that can be used by an A2A(S1)/A3ACPU.

POINTS

(1) If a bit device is processed in units of one word, specify "0" or a multiple of 16 as the head device number.
(Specify "9000" or "9000" + a multiple of 16 as the head number when a special relay M is used.)

(2) Use the control table to specify the block number of the extension file register.

REMARKS

(1) The allowable ranges of device numbers and file register capacities vary according to the type of PC CPU used.The User's Manual of the appropriate PC CPU gives details.

(2) Special relays M9000 to M9255 and special registers D9000 to D9255 are divided into three categories:
Read-only; Write-only; System-used.
If data is written to a non-write area, an error can occur in the PC CPU.The ACPU programming manual (Common Instructions) gives details about special relays and special registers.

4.2.6 Read/Write data storage areas

This section describes how to store/set data when data is read/written from/to a PC CPU device. An integer variable, array/character variable, and array can be used as data storage areas. If an array is used, define the array by using the DIM command.

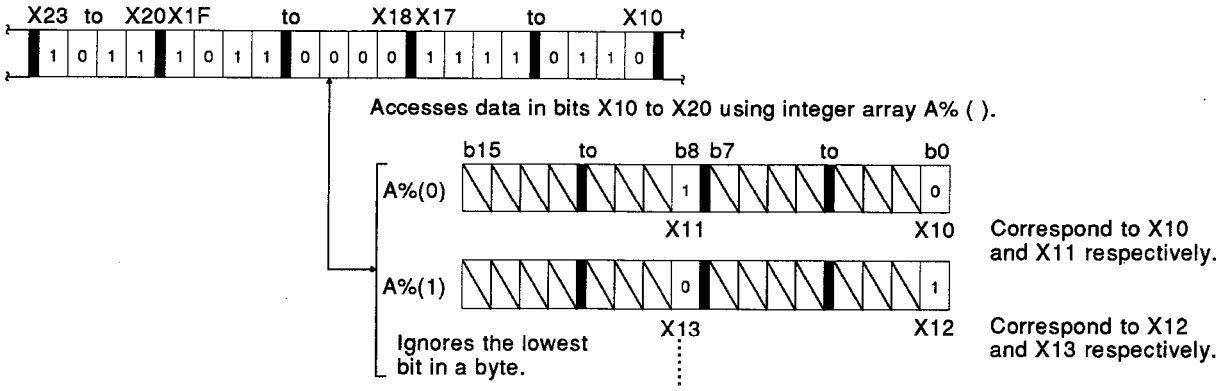
(1) When an integer variable or integer array is used as the data storage area

(a) Bit devices

Data can be handled in units of bit or word. The difference between them is shown below:

- Units of bit

When data is handled in units of bit, ON/OFF data for each bit of variables is stored/set to the lowest bits in each specified variable.
When the corresponding device is OFF, the status of the lowest bit is "0".
When the corresponding device is ON, the status of the lowest bit is "1".

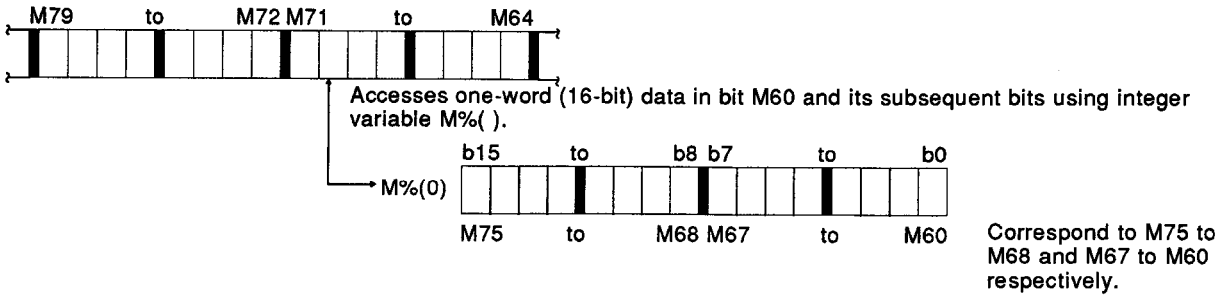


POINT

When an integer array is specified, the data in element 0 is stored.Data in other elements cannot be stored.

- Units of word

  When data is handled in units of word, ON/OFF data for 16 bits per variable is stored/set in ascending order.
  When the corresponding device is OFF, the status of the lowest bit is "0".
  When the corresponding device is ON, the status of the lowest bit is "1".



POINT

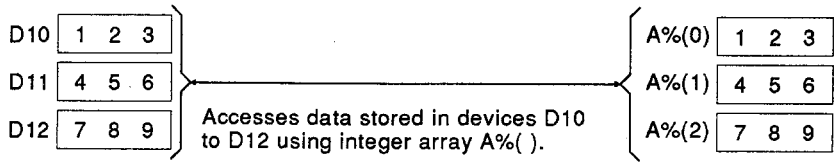
When data in a bit device is handled in units of bit, use the following commands:

(1) If data stored in an integer array is referred to: Use the RDSET command.

(2) If data is set to an integer array: Use the WTSET command.

(b) Word devices

When data is handled in units of word, device number data (one bit per variable) is stored and set.



POINT

When an integer array is specified, the data in element 0 is stored.Data in other elements cannot be stored.

(2) When a character variable or character array variable is used as the data storage area

(a) Bit device

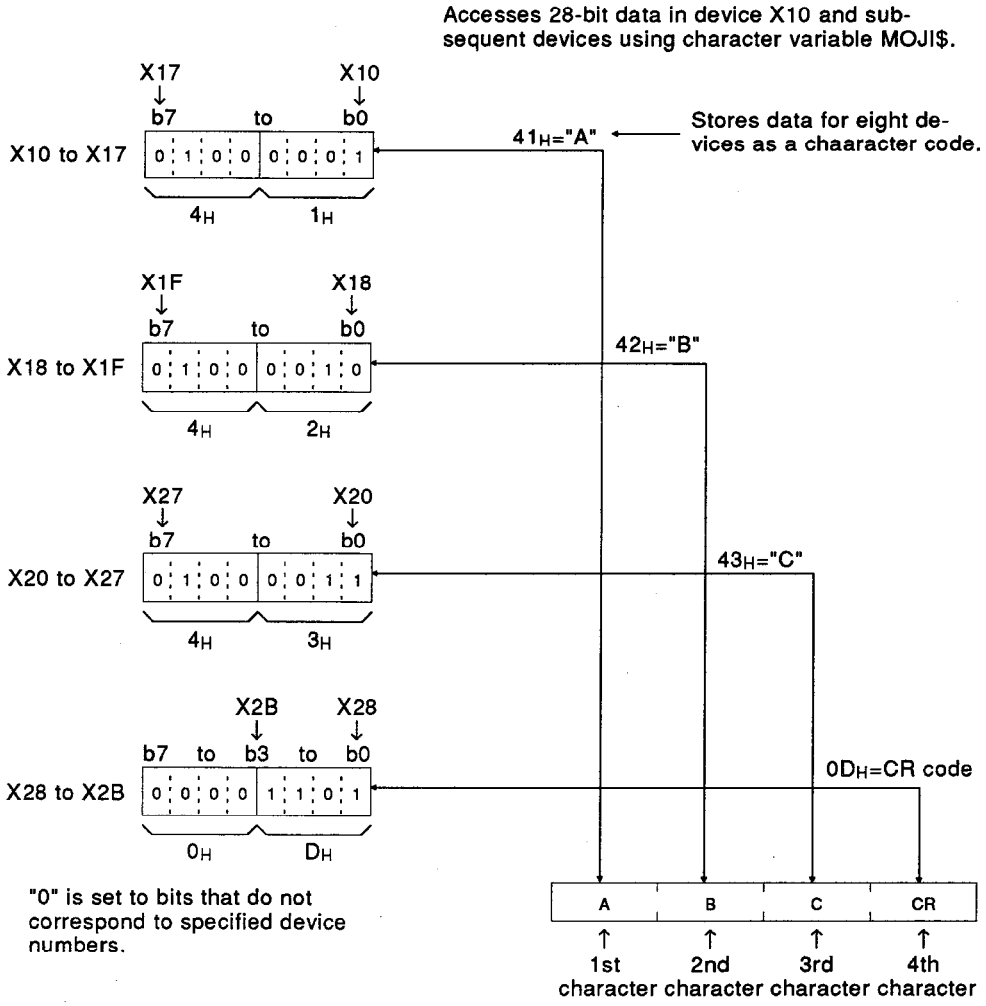
As with item (1), device data can be handled in units of bit or word.

- Units of bit

When read data is handled in units of bit, ON/OFF data for eight bits is regarded as a kind of code and the data for the specified number of bits is stored as characters corresponding to the codes.
Thus, the amount of character data required to store/set data for specified number of bytes is shown below:

Character data = Number of processed bits/8
(decimal part is rounded off)

When the corresponding device is OFF, the status of the lowest bit is "0".
When the corresponding device is ON, the status of the lowest bit is "1".

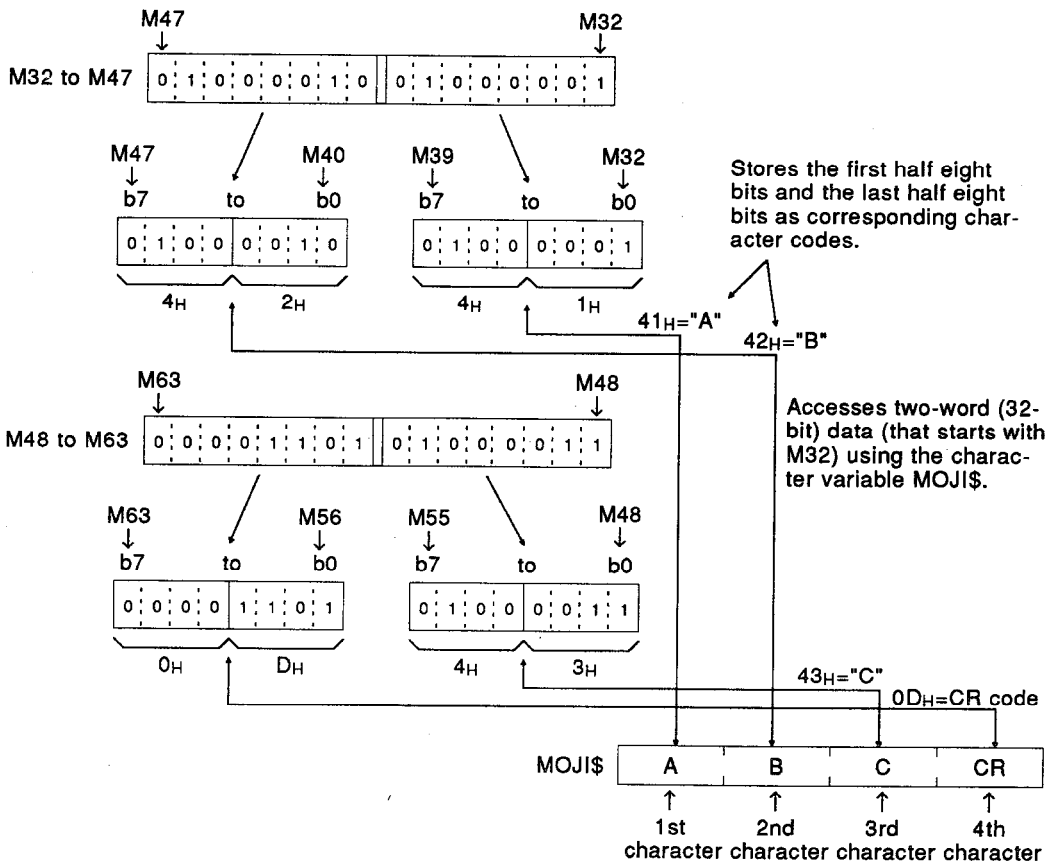


- Units of word

When data read in units of word is specified, data is processed by using 16 serial devices.
Read device numbers for 16 bits are divided into the first half eight bits and the last half eight bits.
These two records of 8-bit data are stored as corresponding character codes.
Thus, the amount of character data required to store/set data for a specified number of bytes is shown below:

Character data = Number of processed bits x 2 (bytes)

When the corresponding device is OFF, the status of the lowest bit is "0".
When the corresponding device is ON, the status of the lowest bit is "1".



When data is read using the PCRD command, if the data contains too many characters to store the data using the variable specified for reading, the data will be stored as shown below:

Assume RD$ is used as a variable for reading:

RD$ .M.I.T.S.U.B.I.S.H.I

Read data .A.D.5.1.H. — Executes

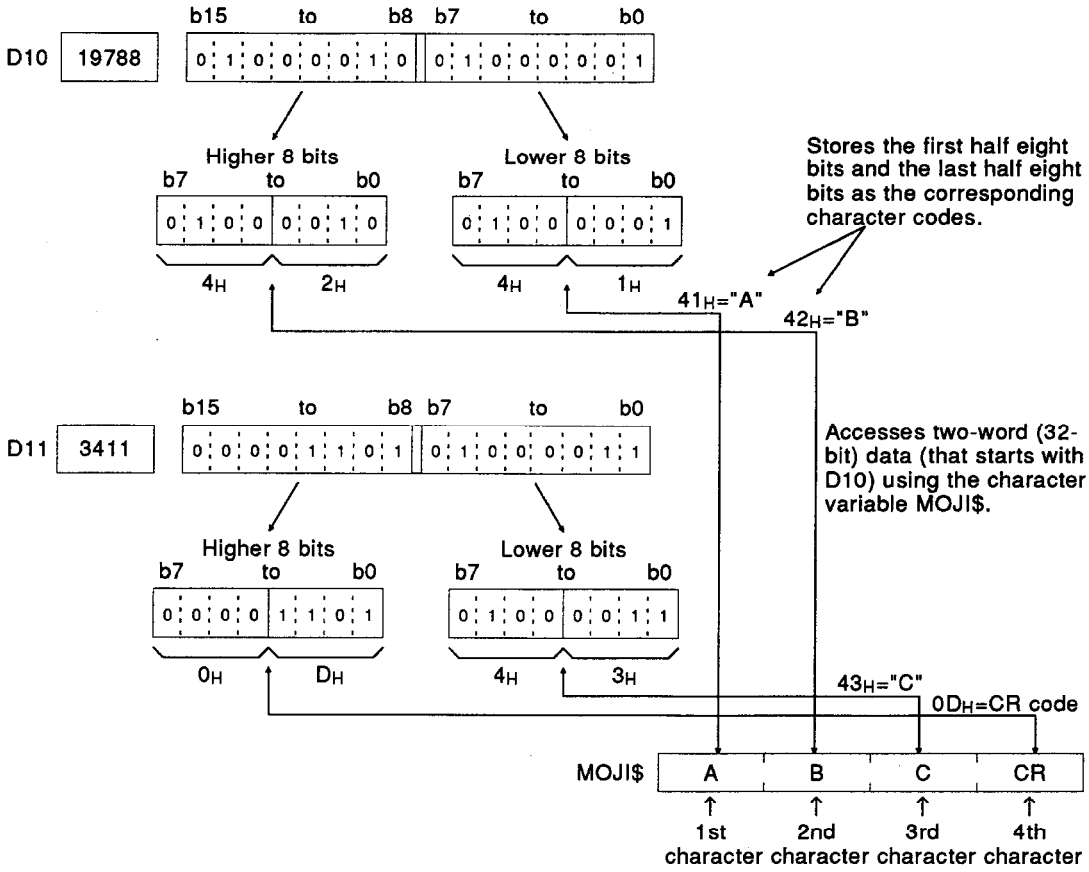
RD$ .A.D.5.1.H.B.I.S.H.I

Stores the data before executing reading.

(b) Word device

When data read in units of word is specified, the read device number for 16 bits are divided into the first half eight bits and the last half eight bits.Those two records of 8-bit data are stored as corresponding character codes.
Therefore, since one-word device data is stored as two words, the amount of character data required to store/set data for specified number of bytes is shown below:

Character data = Number of processed bits x 2 (bytes)



When data is read using the PCRD command, if the data contains too many characters to store the data using the variable specified for reading, the data will be stored as shown below:

Assume RD$ is used as a variable for reading:

RD$ .M.I.T.S.U.B.I.S.H.I

Read data .A.D.5.1.H. — Executes

RD$ .A.D.5.1.H.B.I.S.H.I

Stores the data before executing reading.

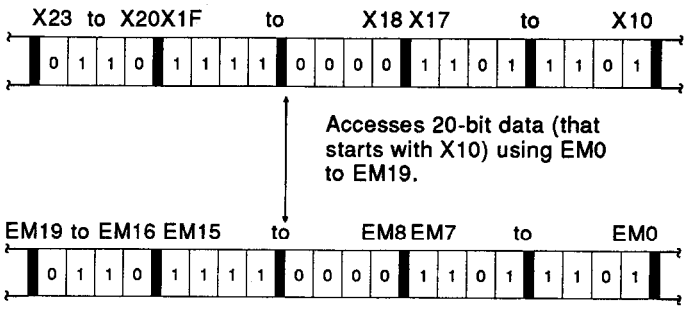
(3) When an internal device (ED or EM) is used as the data storage area

(a) Bit device

Device data can be handled in units of bit or word as shown below:

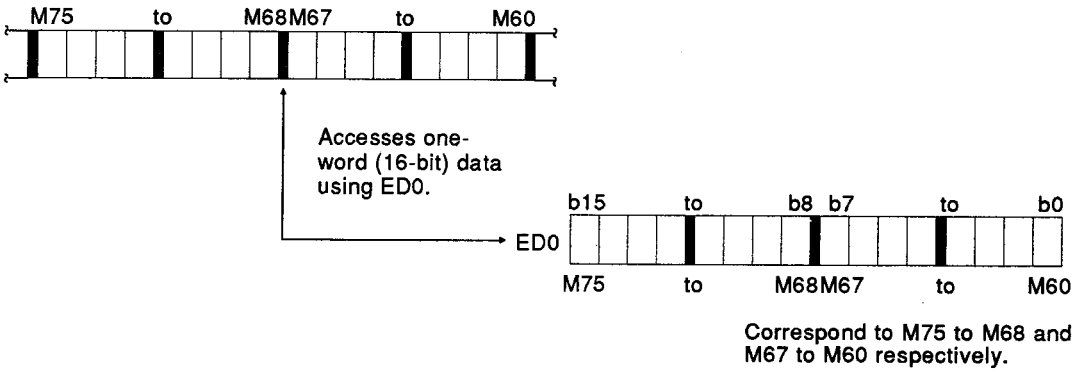
- Units of bit

When data is stored in units of bit using the internal bit device EM, specified devices corresponded to EM bit by bit, and data for the specified bits is stored and set.
When the corresponding device is OFF, the status of the lowest bit is "0".
When the corresponding device is ON, the status of the lowest bit is "1".



- Units of word

When data is stored in units of word using the internal word device ED, ON/OFF statuses of 16 bits per ED are stored to the corresponding bits in ascending order.
When the corresponding device is OFF, the status of the lowest bit is "0".
When the corresponding device is ON, the status of the lowest bit is "1".

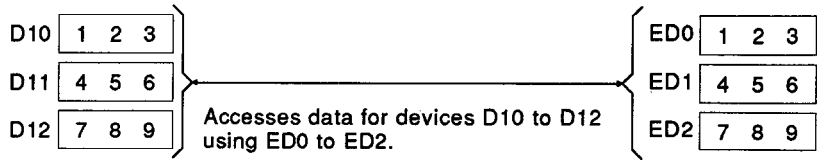


REMARK

Access refers to such input/output processing as data read/write.

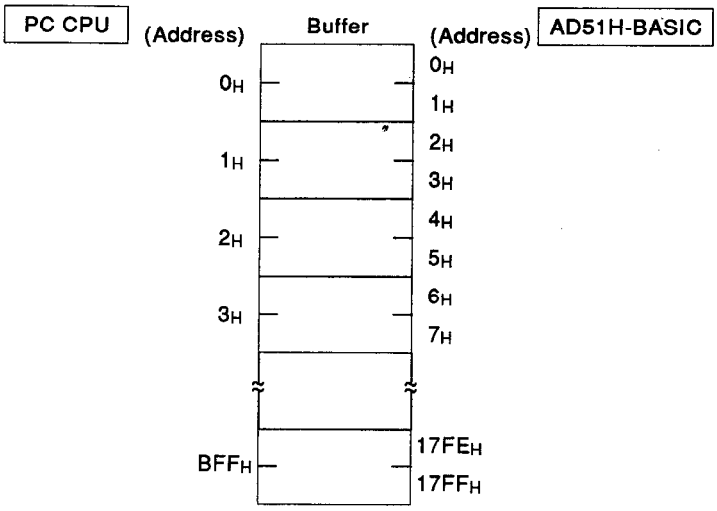
(b) Word device

When data is read in units of word using internal word device ED, data for the one-word device (ED) is stored and set.



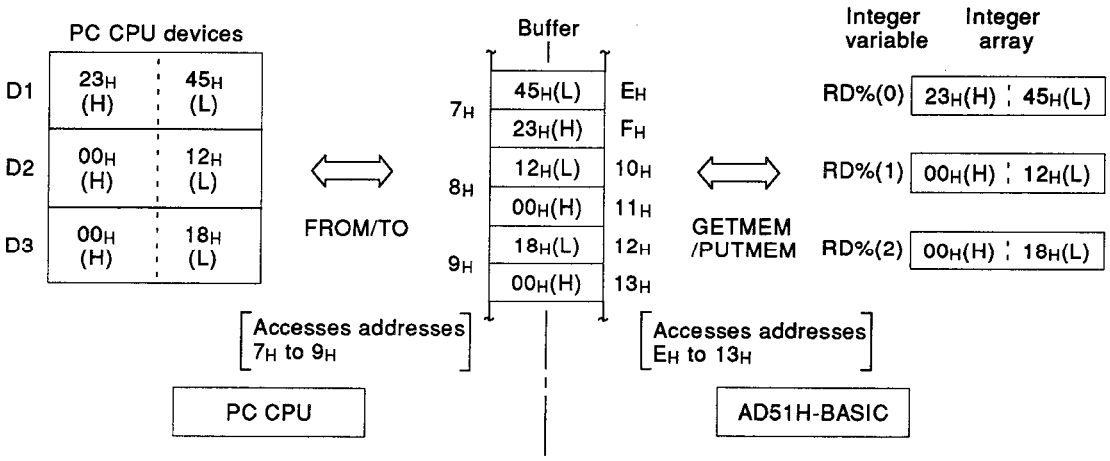
4.3 Data Communications with the Buffer

The AD51H has a 6K byte memory to read/write using the FROM and TO instructions.Using this area, a great deal of data can be communicated to/from a PC CPU. When the buffer is accessed by a BASIC program, use the PUTMEM and GETMEM commands.
In the buffer, the address numbers specified by the PC CPU do not correspond to the same address numbers specified by the AD51H-BASIC as shown below:



In a PC CPU, an address consists of a lead. In a BASIC program, an address consists of a byte.Therefore, one-word and two-word data in a PC CPU requires two addresses and four addresses respectively.
Therefore, BASIC program addresses must always be carefully specified.
Normally, data is communicated as shown below:

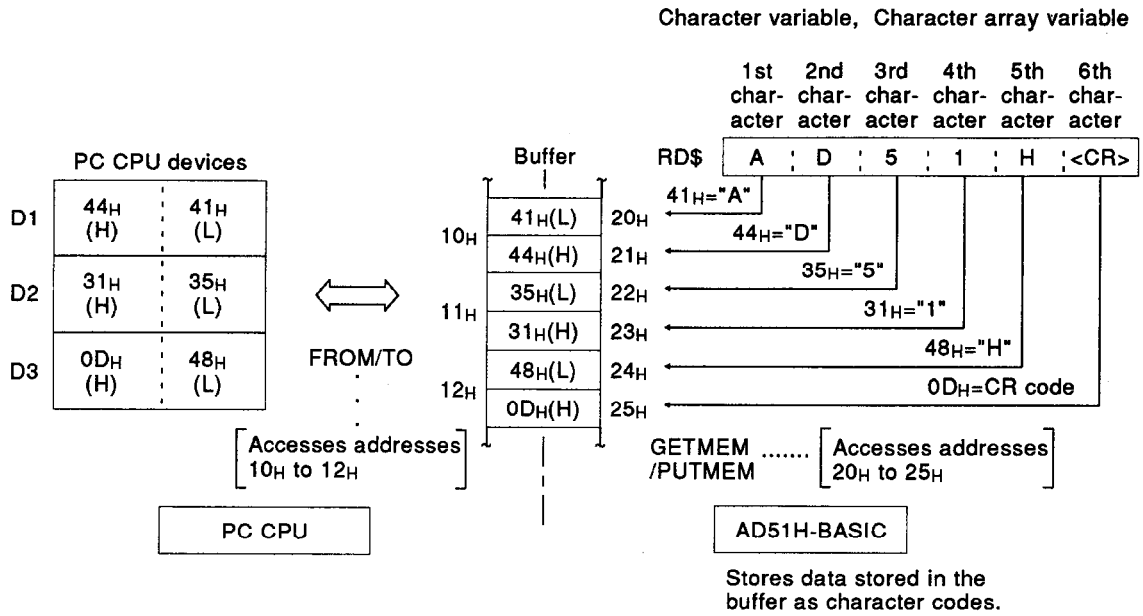
- When numerical data is communicated:



REMARK

The ACPU Programming Manual (Common Instruction) gives details about FROM and TO instructions.

- When character data is communicated:

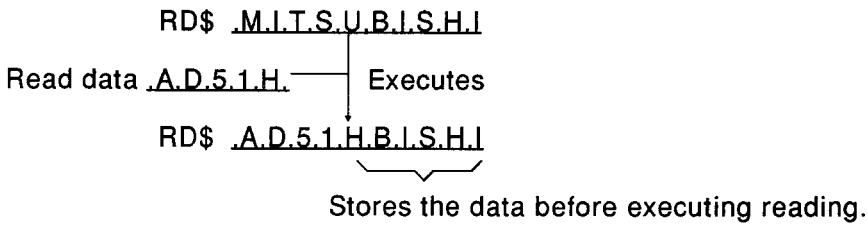


POINT

The PUTMEM and GETMEM commands are shared when the common memory and the internal device (ED) are accessed.If addresses 1800H to 7FFFH are used in a BASIC program, data will be processed using the internal devices.

- When data is read using the GETMEM command, if the data contains too many characters to store the data using the variable specified for reading, the data will be stored as shown below:

  Assume RD$ is used as a variable for reading:

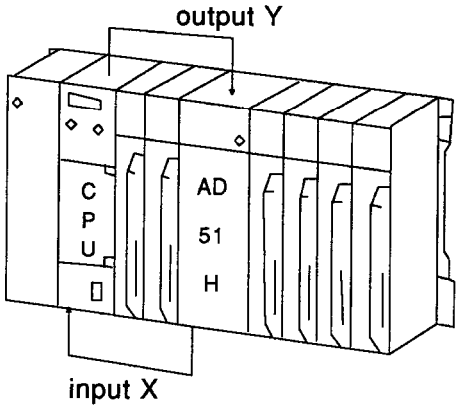


REMARK

Access refers to such input/output processing as data read/write.

5. TRANSFERRING WITH GENERAL USE I/O

The AD51H can transfer data to and from the PC CPU via the input/output devices (Input Device X ..... 32 points, Output Device Y ..... 48 points).

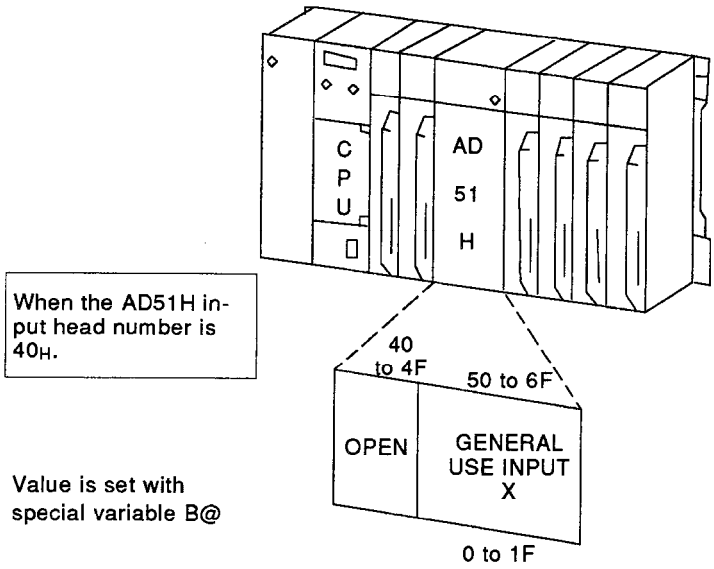


5.1 AD51H to PC CPU (Input Device X)

Input device X can be set ON/OFF from the sequence program with the special variable (B@).

When input device X is set ON/OFF with special variable (B@), the following corresponding input devices in the sequence program are set ON/OFF:

Example



POINT

The device number for input device X which is set with special variable B@ does not depend on the slot in which the AD51H is installed. It is always in the range 0 to 1FH.

The following section illustrates how input device X is seen from the PC CPU whenthe AD51H header address is inserted in the « ».

Sequence Program Input Device X	Signal Name	Value Set with Special Variable B@
X00 + « » B to X0F + « »	Cannot be used.	
X10 + « » to X1A + « »	Can be used for general input.	&H00 to H0A
* X1B + « »	Start multitask execution	&H0B
* X1C + « »	Abort multitask execution	&H0C
* X1D + « »	AD51H system down	&H0D
X1E + « » · X1F + « »	Reserved for system use (cannot be used).	
X20 + « » to X2F + « »	Can be used for general input.	&H10 to &H1

* X1B +« », X1C+« », and X1D+« » are set ON/OFF by the system. The user should not set them ON/OFF him/herself.

X1E+« » and X1F+« » are used by the system, and cannot be used by the user. If, for some reason, these device numbers are used (set ON/OFF) by the sequence program, the functionality of the AD51H cannot be guaranteed.

POINT

If the AD51H head address is set to 60H, the input device numbers as seen from the PC CPU are as follows:

X00+« » to X0F+« » → X60 to X6F

X10+« » to X1A+« » → X70 to X7F

X1B+« » → X7B

X1C+« » → X7C

X1D+« » → X7D

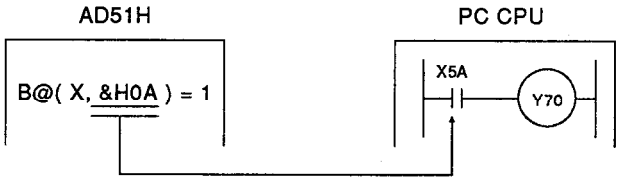
X1E+« » · X1F+« » → X7E · X7F

X20+« » to X2F+« » → X80 to X8F

- X10+« » to X1A+ · X20+« » to X2F+« » ..... General use input

  Sequence program input device X is controlled by the AD51H special variable B@.

  For example, if the head address is 40_H...



POINT

If &H0A is set to 1 using AD51H special variable B@, X5A in the sequence program is set ON.

- X1B + « » ..... Start multitask execution

  When initial setup has been completed and BASIC is ready to start multitask execution,this signal is turned on. If either "abort multitask execution" (X1C+« ») or "AD51H system down"(X1D+« ») is ON, "start multitask execution" (X1B+« ») is OFF.

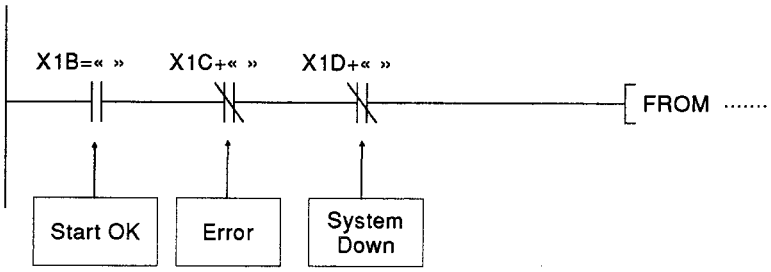
  X1C + « » ..... Abort multitask execution

  If an error occurs, and error management is not performed before returning from execution, this signal is turned on.

  X1D + « » ..... AD51H system down

  If an error occurs which forces the AD51H system to shut itself down, this signal is turned on.

  When the AD51H is in an anomalous condition, these three inputs can be used to cause an interlock in order to abort the operation of the commands FROM, TO, etc. and of the I/O devices.



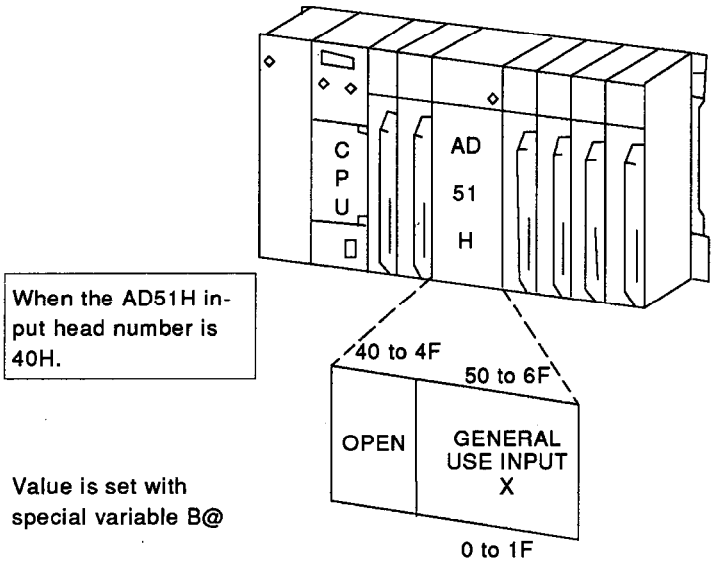
For more details, refer to the AD51H/AD51H-S3 Users' Manual.

5.2 PC CPU to AD51H (Output Device Y)

The value of special variable B@ is controlled by the sequence program with output device Y.

When special output device Y is set ON/OFF by the sequence program, the following corresponding AD51H output devices are set ON/OFF:

Example



POINT

The device number for output device Y which is set with special variable B@ does not depend on the slot in which the AD51H is installed. It is always in the range 0 to $1F_H$.

The following section illustrates how output device Y is seen from the PC CPU when the AD51H header address is inserted in the « ».

Sequence Program Input Device X	Signal Name	Value Set with Special Variable B@
Y00+« » to Y0F+« »	Not yet used. (However, it can be used in the sequence programas a substitute for internal relay .)	
Y10+« » to Y20+« »	Can be used for general output.	&H00 to &H10
Y21+« » to Y28+« »	Specifies program to start. (BASIC tasks no. 1 to 8)	&H11 to &H18
Y29+« »	Program start	&H19
Y2A+« » to Y2F+« »	Reserved for system use (cannot be used).	&H1A to &H1F

Y2A+« » to Y2F+« » is used by the system, and cannot be used by the user. If, for some reason, these device numbers are used (set ON/OFF) by the sequence program, the functionality of the AD51H cannot be guaranteed.

POINT

If the AD51H head address is set to 60H, the output device numbers as seen from the PC CPU are as follows:

Y00+« » to Y0F+« » → Y60 to Y6F

Y10+« » to Y20+« » → Y70 to Y80

Y21+« » to Y28+« » → Y81 to Y88

Y29+« » → Y89

Y2A+« » to Y2F+« » → Y8A to Y8F

- Y00+« » to Y0F+« » ..... Not yet used.
  (However, can be used as a substitute for internal relay .)

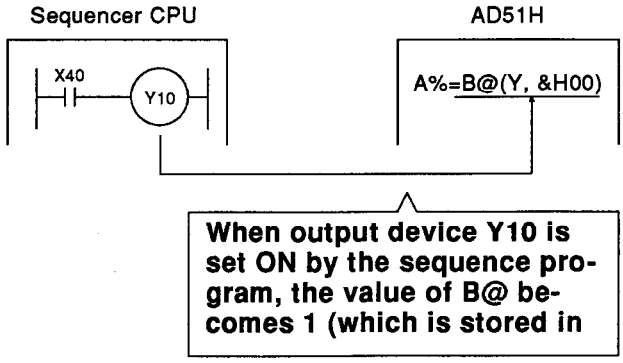
Can be used by the sequence program as a substitute for internal relay .

Cannot be used for transfer with the AD51H.

- Y10+« » to Y20+« » ..... General use output

  The value of special variable B@ is controlled by the sequence program with output device Y.

  For example, if the head address is 00H...



- Y21+ « » to Y28+« » ..... Specifies program to start. (BASIC tasks no. 1to 8)

  Setting a device in the range Y21+« » to Y28+« » ON declares the corresponding program ready to start.

  The following illustrates the output device Y and its corresponding programs.

Y21+« »	BASIC task #1 program is ready to start.
Y22+« »	BASIC task #2 program is ready to start.
Y23+« »	BASIC task #3 program is ready to start.
Y24+« »	BASIC task #4 program is ready to start.
Y25+« »	BASIC task #5 program is ready to start.
Y26+« »	BASIC task #6 program is ready to start.
Y27+« »	BASIC task #7 program is ready to start.
Y28+« »	BASIC task #8 program is ready to start.

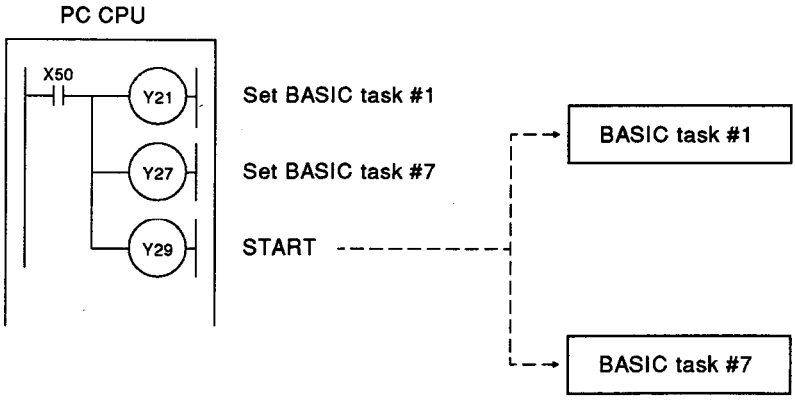
POINT

To start a program which has been specified as ready to start using Y21+« » to Y28+« » it is necessary to set Y29+« »ON.

- Y29+« »Program start (Cannot be used as general purpose output.)

  Starts the programs specified as ready to start using output devices Y21+« » to Y28+« ».

  Example: Start the programs for BASIC tasks #1 and #7.



6. DATA FILE I/O PROCEDURES

There are two types of data files that can be created in BASIC.
These files have the following features.

- Sequential File

  Data is written continuously in order. This makes it possible to save data without creating any wasted space on a memory card or floppy disk, but it makes it impossible to read or write only one part of the file.

POINT

A sequential file is similar to data written on a tape.
Data is written in order from the beginning, but it is not possible to instantly access data in the middle of the file.

- Random File

  Data is written in separate blocks. Therefore, only one part of the file can be read or written. If the data written does not fill an entire block, wasted space is created.

POINT

A random file is similar to data written on a CD.
Data in the middle of the file can be accessed instantly.

REMARK

Sequential file and random file data can be used interchangeably, but the following points should be kept in mind.

(1) If sequential file data is used for a random file, the sequential file data must correspond correctly with a random file field.

(2) If random file data is used for a sequential file, it is necessary to consider what character will be used as a special symbol to separate the data in the sequential file.
For more information on sequential file delimiter symbols, see section 6.2.

6.1 File Numbers

In BASIC, each program can use up to eight data files simultaneously. File numbers are used with data file input/output commands to specify which of these eight files the command is to be performed on.

(1) Correspondence between the Data Files and the File Numbers

A data file is assigned a file number when the file is opened with the OPEN command.

Example

Sequential Files

1) OPEN "0:A-1.DAT" FOR OUTPUT AS #1

When outputting data to sequential file "0:A-1.DAT", use file number 1.

2) OPEN "0:B-10.DAT" FOR INPUT AS #3

When inputting data from sequential file "0:B-10.DAT", use file number 3.

Random Files

3) OPEN "1:R-2.DAT" AS #2

When inputting data from or outputting data to random file "1:R-2.DAT", use file number 2.

(2) Using File Numbers in I/O Commands

Instead of directly specifying the data file to operate on, data file I/O commands specify the file via the file number.
I/O is performed on a file depending on its file number, which is specified with the OPEN command.

Example

1) PRINT #1, "MITSUBISHI"

→ "MITSUBISHI" is output to the sequential file "0:A-1.DAT".

2) INPUT #3, A$

→ Data is input from the sequential file "0:B-10.DAT" to the variable A$.

3) FIELD #2, R$ AS 100

→ The file buffer for random file "1:R-2.DAT" is defined.

6.2 Sequential File I/O Procedures

Below is a summary of I/O procedures for sequential files.

When a sequential file is opened, it must be specified for either reading or writing. The file can then only be used for the type of operation selected (read or write).

(1) Writing Data to the File

1) The file is opened in output mode.

```
OPEN "<FILE NAME>" FOR OUTPUT AS #n
```

(1) n indicates the file number.

(2) For information on specifying a <FILE NAME>, see Appendix 1.

2) Data is written to the file.

```
PRINT #n, <DATA TO WRITE>, .........
PRINT #n, USING  ".......", <DATA TO WRITE>, .......
```

3) The file is closed.

```
CLOSE #n
```

POINT

When finished writing data to the file, always execute the CLOSE command.
If this command is not executed, data may not be written to the file.

To append data at the end of a file that already has data written in it, open the file with the 'OPEN " <FILE NAME>" FOR APPEND AS #n' command. When a file is opened this way, new data is written starting at the end of the data already written in the file. If a file is opened with 'OPEN " <FILE NAME>" FOR OUTPUT AS #n', all data previously written in the file is erased.

(2) Reading Data from the File

1) The file is opened in input mode.

```
OPEN "<FILE NAME>" FOR INPUT AS #n
```

2) Data is read from the file.

```
INPUT #n, <VARIABLE TO STORE DATA>, .......
LINE INPUT #n, <VARIABLE TO STORE DATA>, .......
```

3) The file is closed.

```
CLOSE #n
```

Data that is read from or written to a sequential file is organized as follows.

When Data is Handled as Character Data

Example

A$ = "ABC"

B$ = "123"

C$ = "DEF"

- PRINT #n, A$; B$; C$ ⟶ A B C 1 2 3 D E F [CR] [LF]

Data File Contents

In this case, "ABC", "123", and "DEF" are not separated at all, so the data file contents become like one item of data.

To handle the data as three separate items, write the comma character as shown below.

When writing a comma to a file, be sure to enclose it in quotes.

- PRINT #n, A$;","; B$;","; C$ ⟶ A B C , 1 2 3 , D E F [CR] [LF]

Data File Contents

Since BASIC uses the comma as a delimiter symbol, the above data can be handled as three items.

Example

D$ = "AD51H,A-SERIES"

E$ = "MITSUBISHI"

- PRINT #n, D$; E$

↓

A D 5 1 H , A - S E R I E S M I T U B I S H I [CR] [LF]

In this case, since E$ contains the delimiter character (comma), when the data is read, it is handled as two items as shown below.

"AD51H"
"A-SERIESMITSUBISHI"

The quote character can be used to separate the data in the file the same way as it is separated in the original data.
The quote character must be written using the CHR$(&H22) function.

The CHR$(&H22) function must be used to write the quote character.
" " " is not allowed.

- PRINT #n, CHR$(&H22);D$;CHR$(&H22);",";E$

↓

" A D 5 1 H ; A - S E R I E S " M I T U B I S H I [CR] [LF]

In BASIC, data enclosed in quotes is considered one item. Using quotes, D$ and E$ can be correctly separated in the file.

When Data is Handled as Numeric Data

When numeric value data is written to a sequential file, the data is automatically converted to character data before it is saved. When data is read from a sequential file as numeric values, the character data from the file is automatically converted to numeric values. Therefore, data in sequential files containing only numeric values can be handled as character data or numeric data.

Example

A=80. 5

B=13. 15

- PRINT #n, A; B ⟶ Δ 8 0 . 5 Δ Δ 1 3 . 1 5 Δ [CR] [LF]

Data File Contents

POINT

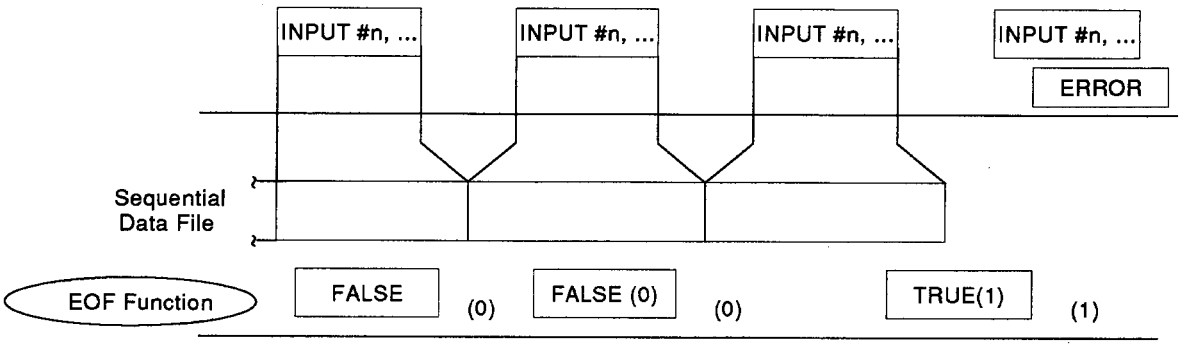
When numeric value data is written, whitespace is automatically inserted just as with the PRINT command for the screen.
For more information on whitespace control, see section 3.10.

In the example above, the values 80.5 and 13.15 are not separated, so in the data file they are considered one data item.
Also, in BASIC, when data is handled as numeric values, whitespace is ignored. However, the data "80.512.15" cannot be handled as one numeric value, so an error occurs.
The comma character is used to separate the data items so that they are handled as two items.

- PRINT #n, A;",";B ⟶ Δ 8 0 . 5 Δ , Δ 1 3 . 1 5 Δ [CR] [LF]

Data File Contents

BASIC uses the comma character as a delimiter symbol, so in the above example, the data can be handled as two separate numeric values.
Sequential file data is read in order from the beginning of the data file. However, if there is no more data left to read, an "Input past end" error occurs.
In order to prevent this error, the EOF function is used. The EOF function returns the value "TRUE" when there is no more data to read. It can be used as a conditional in an IF command.



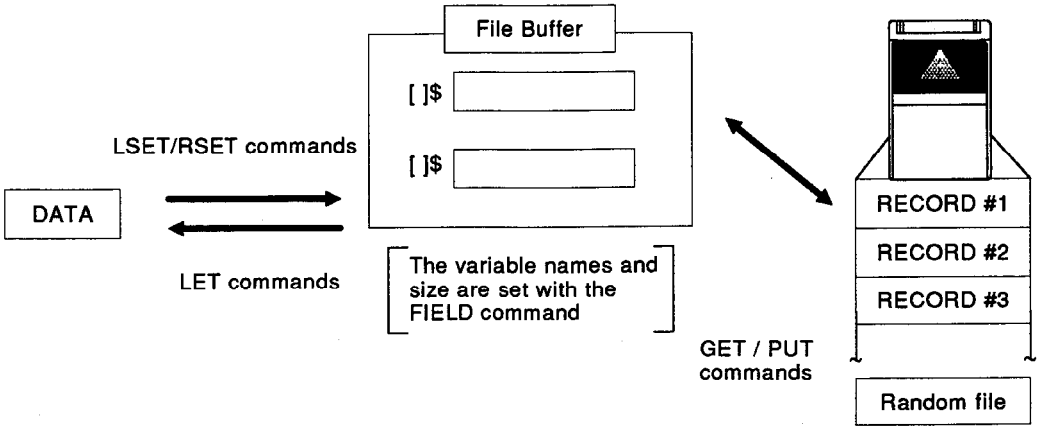
Example

```
1   'Record data to a sequential file.
2   '
10  OPEN "O:TEST.DAT" FOR OUTPUT AS #1:'Create file TEST.DAT and assign
    file number 1
20  INPUT "DATE:";D$
30  LINE INPUT "PRODUCT NAME:";H$
40  INPUT "QUANTITY:";K
50  INPUT "PERCENTAGE(%):";W
60  PRINT #1,D$;",";                    :'Write the data input in line 20
                                         to the file
70  PRINT #1,CHR$(&H22);H$;CHR$(&H22);",";:'Write the data input in line
                                         30 to the file
80  PRINT #1,K;",";                     :'Write the data input in line 40
                                         to the file
90  PRINT #1,USING  "###.#";W           :'Write the data input in line 50
                                         to the file
100  INPUT "Continue? (Y/N)";Y$
110  IF Y$="Y" OR Y$="y" GOTO 20         :'Return to input more data
120  CLOSE #1                            :'Close the file
130  END
```

```
1   'Read data from the file written above.
2   '
10  OPEN "O:TEST.DAT" FOR INTPUT AS #1:'Open file TEST.DAT in input mode
                                         and assign file number 1
20  IF EOF(1) GOTO 100                  :'If the end of the file is
                                         detected, end the program
30  INPUT #1,D$                         :'Read the date from the file
40  INPUT #1,H$                         :'Read the product name from the
                                         file
50  INPUT #1,K                          :'Read the quantity from the file
60  INPUT #1,W                          :'Read the percentage from the file
70  PRINT D$,H$,K;"units",W;"%"         :'Display the data that was read
                                         from the file
80  PRINT
90  GOTO 20
100  CLOSE #1                           :'Close the file
110  END
```

6.3 Random File I/O Procedures

Once a random file is opened, both input and output (read and write) operations can be performed. Random file data passes along the following I/O path.



- Random files are organized into separate records. Each record has a record number which is specified whenever data is read or written. The data is then input from or output to the file buffer. The number associated with a block is called the <record number>.
- The "file buffer" acts as a "window" to the random file when I/O is performed. When performing I/O to the file buffer, character variables that were defined with the FIELD command are used.
  Because of this, all data written to the file buffer must be character data. For example, when writing a numeric value, either the STR$ function or the functions 2shown below can be used to convert the number to a character string. The data is then input to the file buffer using the LSET or RSET command, and written to the file with the PUT command.

Integer Number Data	→	MKI$ (I)	(Becomes 2-byte character data)
Single Precision Real Number Data	→	MKS$ (R)	(Becomes 4-byte character data)
Double Precision Real Number Data	→	MKD$ (D)	(Becomes 8-byte character data)

- When data is written to a random file, it is first stored in the "file buffer" using the LSET and RSET commands. Then the PUT command is used to write the data to the record specified by <record number> in the file.

- When data is read from a random file, the GET command is used to store the data from the block specified by <record number> in the "file buffer". The data is then read into general variables using the LET command.

POINTS

(1) Character variables defined with the FIELD command should not be used with I/O commands such as INPUT or on the left side of the LET command. If they are used in this way, they can no longer be used as the file buffer.

(2) Even if a random file is opened with the intention of creating a new file, the data written in the records on the memory card or floppy disk is not cleared. Please be aware that records that have not been written to yet contain undefined data.

- When numeric data has been converted to character data and written to a file, the VAL function can be used to convert it back to numeric data after reading it in.
  However, numeric data that was converted to character data using the MKI$, MKS$ and MKD$ functions can be converted back to numeric data using the CVI, CVS and CVD functions.
  Character data can be input to the file buffer directly, with no conversion necessary, using the LSET and RSET commands, and then written to the file using the PUT command.
- There are two functions that are useful when random files are being used.

  LOC (file number) ..... Directly assumes the value of the record number of the record last accessed in the specified file with the GET or PUT command.

  LOF (file number) ..... Returns the value of the record number of the last record in the specified file.

  For details on how to use these functions, see the **example**.

POINT

When random files are opened, write/read mode is not specified. Therefore with only one OPEN command and one FIELD command,both read and write operations can be performed.

REMARKS

(1) Each random file may contain up to 32767 records.

(2) If new data is written to a record that already contains data, the current data is overwritten by the new data.

The following is a summary of random file I/O procedures.

1) The file is opened.

OPEN "<FILE NAME>" AS #n

POINTS

(1) n indicates the file number.

(2) For information on how to specify the <FILE NAME>, see Appendix 1.

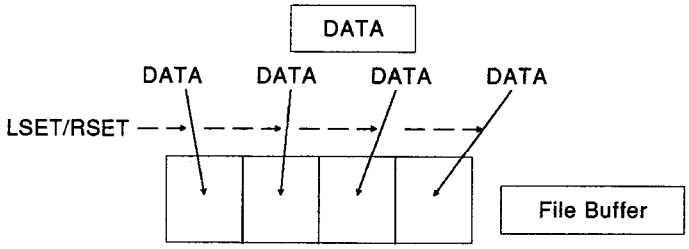
2) The file buffer is defined.

FIELD #n, 8 AS A$, 32 AS B$

3) When writing data to the file, the data is first input to the file buffer.

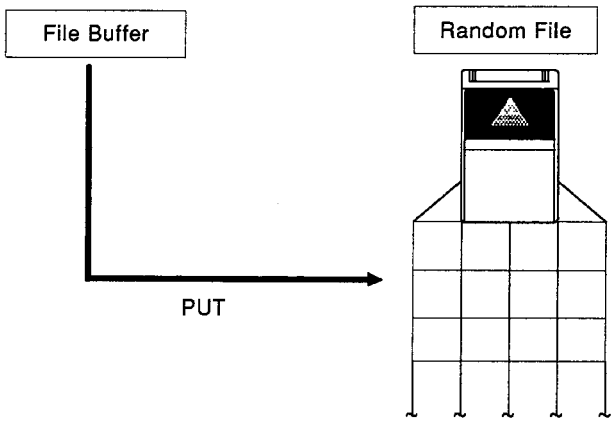
LSET A$ = MKD$ (D#) ..... Numeric value data (D#) is converted to character data with the MKI, MKS and MKD functions.

LSET B$ = X$



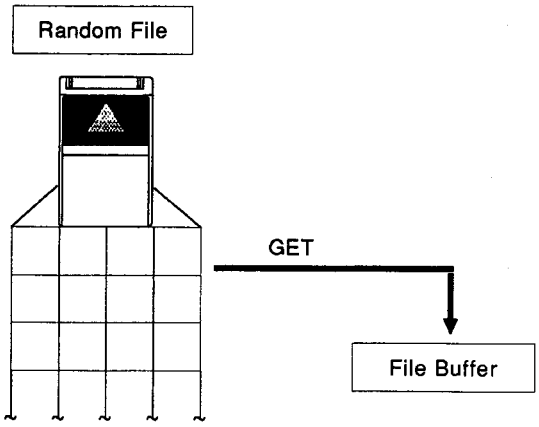
The PUT command writes the data from the file buffer to the record.

PUT #n, <RECORD NUMBER>



3)' When reading data from the file, the GET command reads data from a record into the file buffer.

GET #n, <RECORD NUMBER>



The data in the file buffer is read into variables.

D1# = CVD (A$) ..... Numeric data is converted back into its original numeric type with the CVI, CVS and CVD functions.

X1$ = B$

4) The file is closed.

CLOSE #n

POINTS

(1) When finished inputting and outputting data, always execute the CLOSE command.

(2) Once the CLOSE command has been executed, input/output cannot be performed on the file or the file buffer.

Example

```
1   'Record data to a random file.
2   '
10  OPEN "R-TEST.DAT" AS #1            :'Create file R-TEST.DAT and assign
                                        file #1
20  FIELD  #1,8 AS D$,30 AS H$,2 AS K$,4 AS W$:'Perform file buffer
                                        assignments
30  R=0
40  R=R+1                              :'Specify a record number
50  INPUT "DATE:";D1$
60  LINE INPUT "PRODUCT NAME:";H1$
70  INPUT "QUANTITY:";K1
80  INPUT "PERCENTAGE:";W1
90  LSET D$=D1$                        :'Write the data input in line 50
                                        to the file buffer
100  LSET H$=HI$                       :'Write the data input in line 60
                                        to the file buffer
110  LSET K$=MKI$(K1)                  :'Write the data input in line 70
                                        to the file buffer
120  LSET W$=MKS$(W1)                  :'Write the data input in line 80
                                        to the file buffer
130  PUT #1,R
140  INPUT "Continue? (Y/N)";Y$
150  IF Y$="Y" OR Y$="y" GOTO 40       :'Return to input more data
160  CLOSE #1                          :'Close the file
170  END
```

```
1   'Read data from the random file written above.
2   '
10  OPEN "R-TEST.DAT" AS #1            :'Open file R-TEST.DAT and assign
                                        file number 1
20  FIELD #1,8 AS D$,30 AS H$,2 AS K$,4 AS W$:'Perform file buffer
                                        assignments
30  FOR R=1 TO LOF (1)                 :'Repeat until the end of the random
                                        file (the value returned by the LOF
                                        function)
40  GET #1,R                           :'Read data from the random file
50  D1$=D$:H1$=H$                      :'Read data from the file buffer
60  K=CVI(K$)
70  W=CVS(W$)
80  PRINT D$,H$,K;"units",W;"%"        :'Display the data that was read
                                        from the file
90  PRINT
100  NEXT R
110  CLOSE #1                          :'Close the file
120  END
```

6.4 Data File Handling Dangers

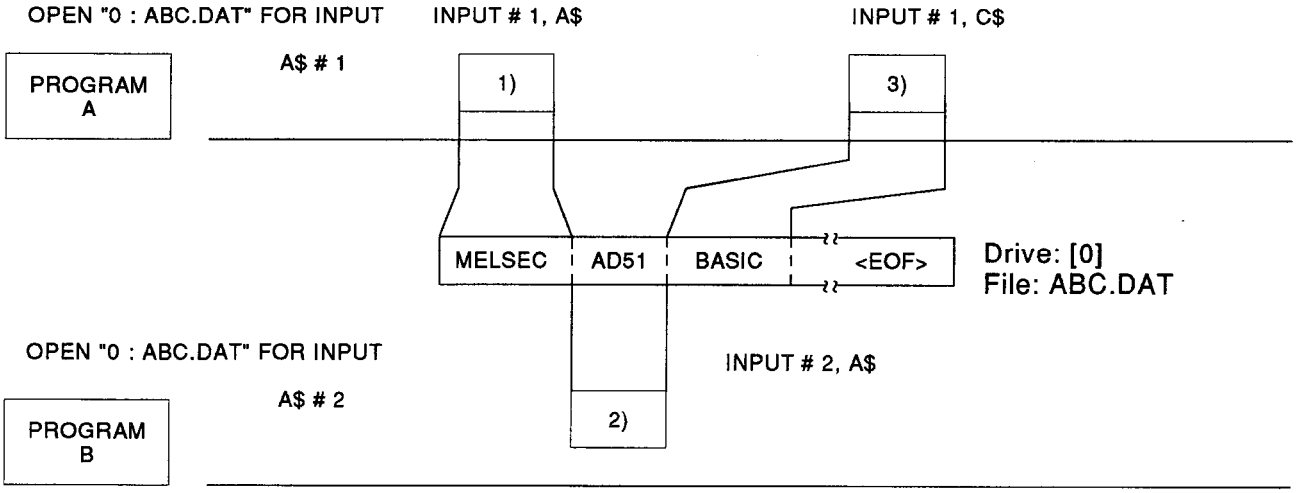
6.4.1 Data File Handling under Multitasking

If several programs running simultaneously under multitasking each try to input data to or output data from the same file at the same time, the procedures explained below are followed.

- Several programs can open the same random file at the same time. In this case, the AD51H operating system automatically performs exclusive access management for each I/O command, so it is not necessary for the user to perform exclusive access management.
- Several programs can open the same sequential file at the same time only if input mode or append mode is specified. In this case, the AD51H operating system automatically performs exclusive access management for each I/O command, so it is not necessary for the user to perform exclusive access management.

Example

Two programs running at the same time try to read from sequential file ABC.DAT on drive 0:



Data is read in order, 1), 2), 3).

"MELSEC" is stored to [program A] variable A$

"AD51H" is stored to [program B] variable B$

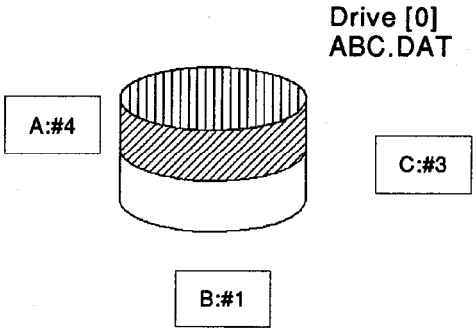
"BASIC" is stored to [program A] variable A$

If either program A or program B wanted to read all of the data in the file, exclusive access management must be performed using the ZRESERVE and ZRELEASE commands. For more information on exclusive access management, see section 8.2.

POINT

If either program opens a file in output mode, exclusive access management must be performed using the ZRESERVE and ZRELEASE commands. If exclusive access management is not performed, an error will occur when a second program tries to open the same file. For more information on exclusive access management, see section 8.2.

- The file number specified when a file is opened is managed by each program separately.Therefore, the file number selected when a data file is opened simultaneously by several programs can be different in each program.

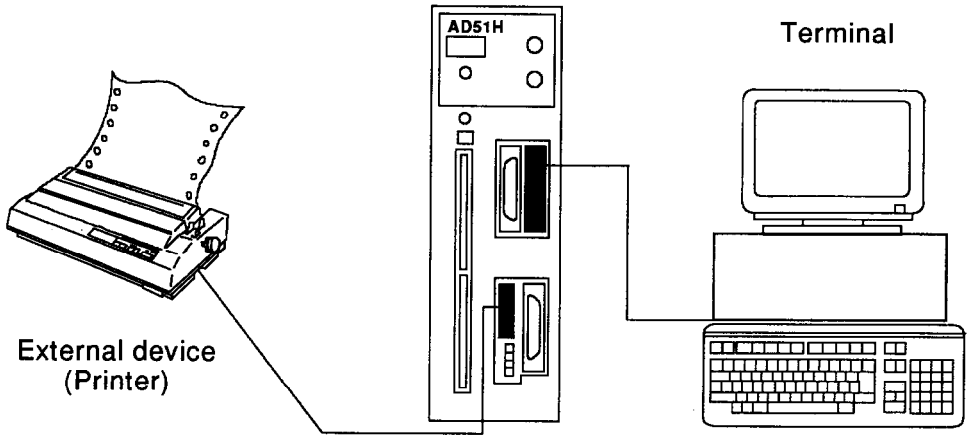


6.4.2 The Number of Files that can be Handled by Each Program

One program can open a maximum of eight files at the same time.
Also, under multitasking, each program can have eight files open at the same time, so the total number of files that can be open simultaneously for the entire system is 64

7. COMMUNICATING WITH EXTERNAL DEVICES

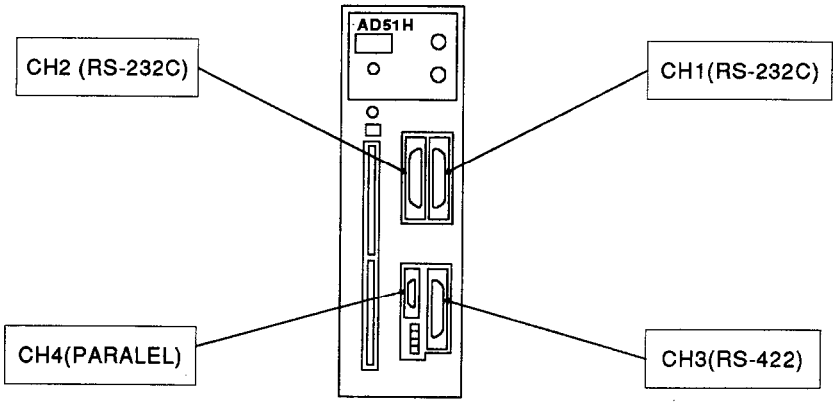
The AD51H is equipped with a great many communications interfaces.
The AD51H communicates with external devices and terminals via these interfaces.



> If the AD51H dip switches are used to specify an interface allocated to the console, that interface cannot be used for data communications with an external device (like those described in this section).

7.1 Correspondence of Interfaces to Channel Numbers

Channel numbers are allocated to each interface. None of these channel numbers may be duplicated.
When data communications is done with an external device or a terminal, specify which interface is used by setting its corresponding channel number.



REMARK

The AD51H usually does data communications with an external device or terminal in the no-protocol mode.
If the AD51H does data communications in the protocol mode, a program to support the protocol control must be created.

7.2 Preparations for Communicating Data

When the AD51H communicates data with an external device or terminal, various settings are needed.
This section tells how to do the necessary preparations for communicating data.

7.2.1 Setting communications parameters

When data is communicated via the RS-232 (corresponds to channels 1 or 2) or RS-422 (corresponds to channel 3) interfaces, set the following communications parameters beforehand, using the ZOPEN and ZCNIL commands.
These parameters must match the communicating device.
If these parameters do not match the communicating device, either (a) data will not be communicated, or (b) a communications error may occur.

POINT

It is not necessary to set a communications parameter for the parallel interface (corresponds to channel 4).

- Baud rate ..... Set using the ZOPEN command.

  Specifies how fast data is communicated. In a BASIC program, any of the following speeds can be selected:

  1200 Bps; 2400 Bps; 4800 Bps; 9600 Bps; or 19200 Bps.

- Character length ..... Set using the ZOPEN command.

  Specifies the number of bits the code uses to express communications characters.
  In a BASIC program, one of the following types can be selected:

  - 5-bit ..... Can express 32 characters ranging from 0 to $1F_H$.
  - 6-bit ..... Can express 64 characters ranging from 0 to $3F_H$.
  - 7-bit ..... Can express 128 characters ranging from 0 to $7F_H$.
  - 8-bit ..... Can express 256 characters ranging from 0 to FF_H.

- Parity bit ..... Set using the ZOPEN command.

  Specifies how to check the accuracy of the data (parity check) being communicated.
  In a BASIC program, one of the following settings can be selected:
  - Does not check data.
  - Checks data in even parity.
  - Checks data in odd parity.

REMARK

Parameter refers to the data which the program needs to do data communications.

- Stop bit ..... Set using the ZOPEN command.

  Specifies how to express the data end.
  In a BASIC program, one of the following settings can be selected:
  - 1-bit ..... Expresses the data end using 1-bit data.
  - 2-bit ..... Expresses the data end using 2-bit data.
  - 1.5 bit ..... Expresses the data end using 1.5-bit data.

The following items are set when necessary: Appendix 6 gives details about DC1/DC3 controls and signal control.

- Size of the receive buffer ..... Set using the ZCNTL command.

  The receive buffer automatically stores data received from an external device even if a data-receiving command has not been given.
  If the receive buffer size is set to "0", received data will be ignored if a data-receiving command is not given.

- DC1/DC3 controls ..... Set using the ZCNTL command.

  When DC1/DC3 controls are set to "YES":
  While data is being received, if the receive buffer becomes full, the AD51H will transmit DC3 code;and if the buffer has enough space, DC1 code will be transmitted.
  During data transmission, when DC3 code is received, transmission stops. When DC1 code is received, transmission resumes.
  DC1/DC3 control settings must match the external device.

- Signal controls ..... Set using the ZCNTL command.

  The CS/RS (RS-232C/RS-422) signals and ER/DR (ER-232C only) signals are used to control data communications.
  The settings of these signals are as follows:

Signal	When "YES" is selected	When "NO" is selected
CS	Input signal Data is transmitted data when this signal is turned ON.	["NO" cannot be selected in the RS-232C interface.]
RS	Output signal Goes ON when data can be communicated normally.	Data is received independently of the signal status.
DR	Input signal Data is transmitted when this signal is turned ON.	Data is transmitted independently of the signal status.
ER	Output signal Goes ON when data can be communicated normally.	Data is transmitted independently of the signal status.

POINTS

(1) The CS signal (corresponds to the RS-232C interface) is always controlled.

(2) DR and ER signals cannot be used with the RS-422 interface.

The signal settings on the AD51H do not have to match the external device. These settings can be affected by the cable connection status.

- Break character ..... Set using the ZCNTL command.

  When, during data receive, the AD51H receives that data specified as the break character, the AD51H stops receiving data.
  The break character for the AD51H must be the same as that for the communicating device.

7.2.2 Control table

A great many parameters are needed to execute the ZOPEN and ZCNTL commands.
To do so, use a "control table". In a BASIC program, part of the memory is used as a control table to transfer parameters for notifying the AD51H of the operating status.
In a BASIC program, the necessary parameters are stored in an array. Specifying the array name transfers the parameters to the AD51H.

To execute the ZOPEN command, for example, a great many parameters must be set.
However, if a great many parameters follow a command, it is difficult to read a program.
Therefore, it is easier to read the program by using a control table.

Using the control table

```
100P%(0)=9600
110P%(1)=&H108          :'Baud rate
120P%(2)=1              :'Parity & character
130 ZOPEN #1, P%( )     :'Stop bit
```

7.3 Procedure for Communicating with an External Device

This section shows the procedure for communicating with such external devices as a console, terminal, and printer.

7.3.1 Communications with a console

The following tells how to communicate with a console connected to the AD51H RS-232C/RS-422 interface:

The console settings must be done using the AD51H dip switches. The AD51H/AD51H-S3 User's Manual (Hardware) gives details.

(1) When the console screen is used

1) Use the ZODV command to specify the console as the output device.

ZODV 0 → Must be "0" when the console is specified.

2) Use one of the following commands to output to the console screen and control the screen.

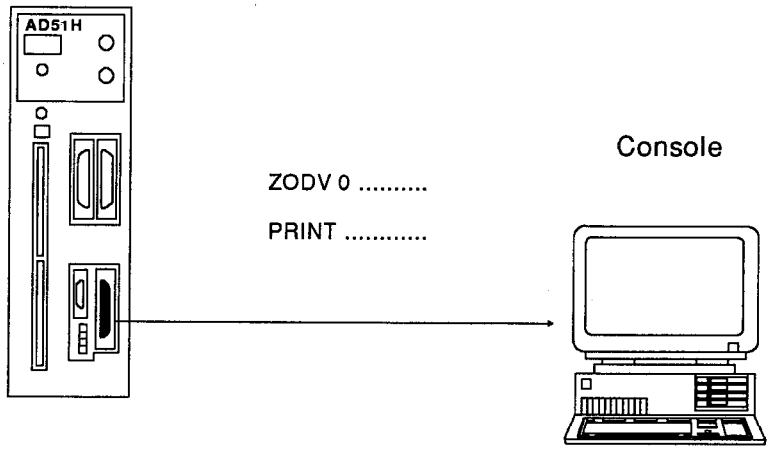
PRINT ..... Displays data on the screen of a specified console.

PRINT USING ..... Displays data on the screen of a specified console (format can be specified).

LOCATE ..... Specifies where data is displayed on the screen of a specified console.

CLS ..... Clears the screen of a specified console.

The PRINT command can be used along with the TAB and SPC functions.



(2) When data is input from a keyboard connected to a console

1) Use the ZIDV command to specify the console as the input device.

ZIDV 0 → Must be "0" when the console is specified.

2) Use one of the following commands to input from the keyboard connected to the console:
However, remember that each command (or function) has several unusable character codes as shown below:

INPUT ..... Cannot include character codes 0 to $1F_H$ (control codes), $2C_H$ (comma), $7F_H$ ([Delete]), 80_H, and FD_H to FF_H in the data.

LINE INPUT ..... Cannot include character codes 0 to $1F_H$ (control codes), $7F_H$([Delete]), 80_H, and FD_H to FF_H in the data.

INKEY$: / INPUT$ ..... (Mathematical expression): Cannot include character codes 0_H, 03_H, 13_H, 80_H, and FD_H to FF_H (control codes) in the data.

7.3.2 Communicating with a terminal

This section shows the procedure for communicating with a terminal connected to an AD51H RS-232C/RS-422 interface.

> If the AD51H dip switches are used to specify an interface allocated to the console, the procedure in this section cannot be applied to a terminal connected to that interface.

(1) When the terminal screen is used

1) Open the interface, and set the parameters so that the utilized terminal is matched.

   ZOPEN <Channel Number>, <Table Variable Name>

   ZCNTL <Channel Number>, <Table Variable Name>

2) Specify the channel number of the interface (to which the utilized terminal is connected) using the ZODV command.

   ZODV Channel Number

3) Use one of the following commands to output data to the terminal screen and to control the screen.

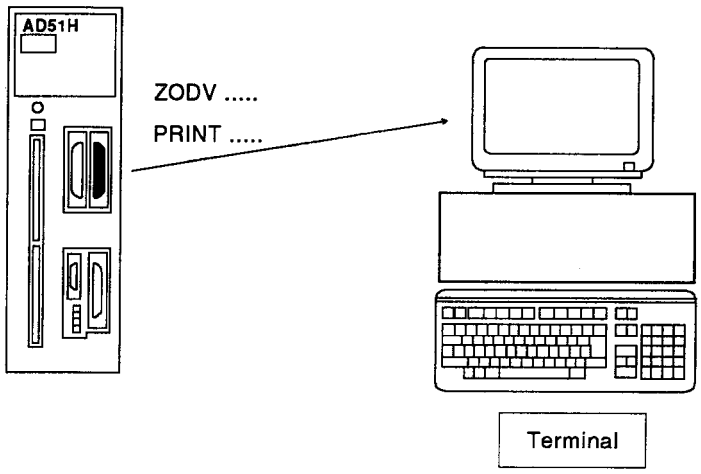
   PRINT ..... Displays data on the screen of a specified terminal.

   PRINT USING ..... Displays data on the screen of a specified terminal (format can be specified).

   LOCATE ..... Specifies where data is displayed on the screen of a specified terminal.

   CLS ..... Clears the screen of a specified terminal.

   The PRINT command can be used along with the TAB and SPC functions.



4) To end the screen display, close the interface, and set the output device to the console again.

ZCLOSE <Channel Number>

ZODV 0

POINTS

(1) It is easier to switch the terminal using the ZODV and ZIDV commands than using the ZSEND and ZRECEIVE commands.

(2) This is because the key codes that are used for controlling the terminal cursor and function keys are automatically converted to be consistent with the AD51H.

(2) When data is input from a keyboard connected to a terminal

1) Open the interface, and set the parameters so that the utilized terminal is matched.

ZOPEN <Channel Number>, <Table Variable Name>

ZCNTL <Channel Number>, <Table Variable Name>

2) Use the ZIDV command to specify the terminal (to which the keyboard is connected) as the input device.

ZIDV <Channel Number>

3) Use one of the following commands to input from the keyboard connected to the terminal.
However, remember that each command (or function) has several unusable character codes as shown below:

INPUT ..... Cannot include character codes 0 to 1F$_H$ (control codes), 2C$_H$ (comma), and 7F$_H$ ([Delete]) in the data.

LINE INPUT ..... Cannot include character codes 0 to 1F$_H$ (control codes) and 7F$_H$([Delete]) in the data.

INKEY$
INPUT$
(Mathematical expression) ..... Cannot include character code 0$_H$ in the data.

4) To end the input operation, close the interface, and set the input device to the console again.

ZCLOSE <Channel Number>

ZODV 0

POINT

Precautions when data is input from a keyboard connected to a terminal:

(1) The program cannot be stopped using the [Break] / [Ctrl] + [C] keys when data is input from a device other than the console using the ZIDV command.
When the INKEY$ or INPUT$ function is used, the 03_H code is input.

(2) When DC1/DC3 is controlled via the interface (to which the terminal is connected), if a code corresponding to DC1 or DC3 is input, the DC1/DC3 control function will prevent data input from the terminal.

(3) The [Esc] key and the key corresponding to the FD_H code on the terminal cannot be used.

If these keys are used, data cannot be correctly input.

7.3.3 Output to a printer

This section gives the procedure to output to a printer connected to the AD51H RS-232C/RS-422/parallel interfaces.

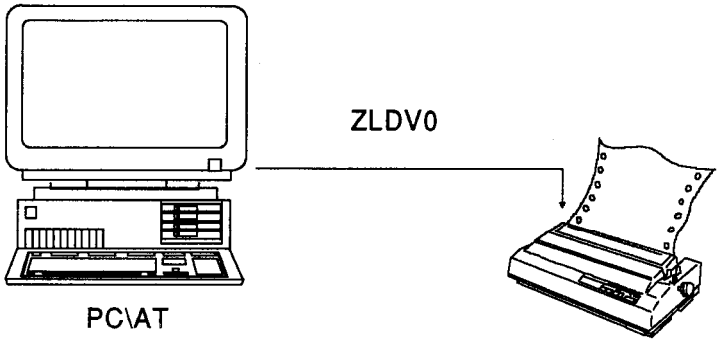
> If the AD51H dip switches are used to specify an interface allocated to the console, the procedure in this section cannot be applied to a printer connected with that interface.

(1) When data is output to a terminal connected to the PC/AT (used as the console)

1) Use the ZLDV command to specify the printer (connected to the PC/AT) as the output device.

ZLDV 0

Must be "0" when the printer connected to the PC/AT is used.



2) Use one of the following commands to output data to the printer.

LPRINT ..... Outputs data to the specified printer.

LPRINT USING ..... Outputs data to the specified printer (format can be specified).

LLIST ..... Outputs a program list to the specified printer.

The LPRINT command can be used along with the TAB and SPC functions.

(2) Output to a printer connected to the AD51H interface

1) Open the interface, and set the parameters so that the utilized terminal is matched.

ZOPEN <Channel Number>, <Table Variable Name>

ZCNTL

REMARK

When the parallel inter face is used, communication parameters are not set.

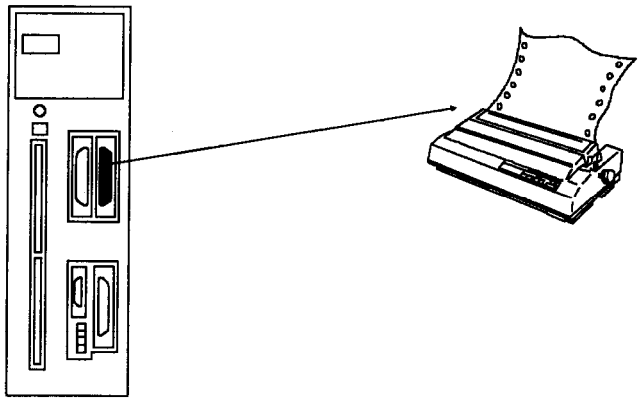
2) Specify the channel number of the interface (to which the used printer is connected) using the ZLDV command.

ZLDV <Channel Number>

3) Use one of the following commands to output data to the printer:

LPRINT	Outputs data to the specified printer.
LPRINT USING	Outputs data to the specified printer (format can be specified).
LLIST	Outputs a program list to the specified printer.

The LPRINT command can be used along with the TAB and SPC functions.



4) To end the printout operation, close the interface.

ZCLOSE <Channel Number>

REMARK

The printer status can be confirmed and the reset signal can be transmitted to the printer by using the ZCNTL command.

The explanation of the ZCNTL command in this manual gives details.

7.3.4 Communicating with other external devices

This section shows the procedures to output/input data to/from a device other than the terminal and/or printer connected to the AD51H RS-232C/RS-422 interfaces.

(1) When data is transmitted to a device ... Use the ZSEND command.

1) Open the interface, and set the parameters so that the utilized external device is matched.

ZOPEN <Channel Number>, <Table Variable Name>

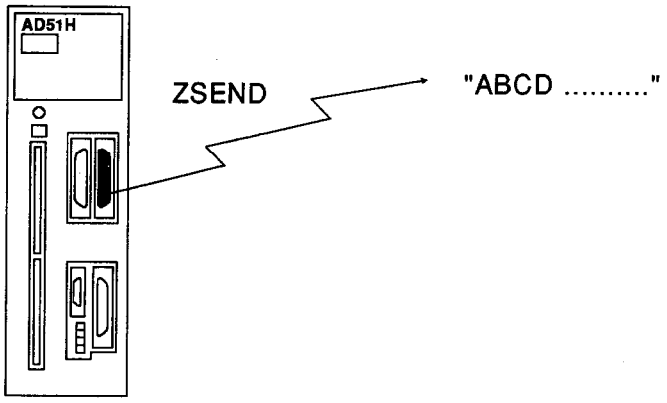
ZCNTL <Channel Number>, <Table Variable Name>

2) Prepare the data to be transmitted. Then, (a) set the number of bytes of send data, and (b) the time-out in the control table (which the ZSEND command uses).

3) Execute the ZSEND command.

ZSEND <Channel Number>, <Table Variable Name>, <Data>

After completing the ZSEND command, the number of bytes of send data is stored in the control table.



4) If data is not transmitted normally, an error occurs. In this case, check the status of the communications line, the interface communications parameters, and the contents of the control table.
An error can have many causes. Carefully check the cable, connecter, parameters, etc.

There are mahy causes of poor communication: the cable, connector, parameters, etc, should each checked one by one.

REMARK

Time-out specification ... If communications is not completed due to an error, this specifies how long communications goes on after the error.

(2) When data is received from a device: Use the ZRECEIVE command or the INPUT$ function.

1) Open the interface, and set the parameters so that the utilized external device is matched.

ZOPEN <Channel Number>, <Table Variable Name>

ZCNTL <Channel Number>, <Table Variable Name>

2) Use the ZRECEIVE command as shown below:

(a) Set (a)the number of bytes of receive data, and (b) the time-out in the control table (which the ZRECEIVE command uses).

(b) Define a variable, integer, or array used for storing the received data as follows:

When the received data is stored in a character variable : [ ]$=SPACE$(255)

When the received data is stored in an integer : [ ]%=0

When the received data is stored in an array : DIM[ ](n)

n indicates the maximum number of array elements used.

(c) Execute the ZRECEIVE command.

ZRECEIVE <Channel Number>, <Table Variable Name>, <Data Storing Variable>

- After completing the ZRECEIVE command, the number of bytes of received data is stored in the control table.
- When, during data receive, the AD51H receives that data specified as the break character, the AD51H stops the receive operation.

2)' Use the INPUT$ function as shown below:

<Received-Data Storing Variable> = INPUT$(<Number of Input Characters>, %<Board Number>, <Time-out>)

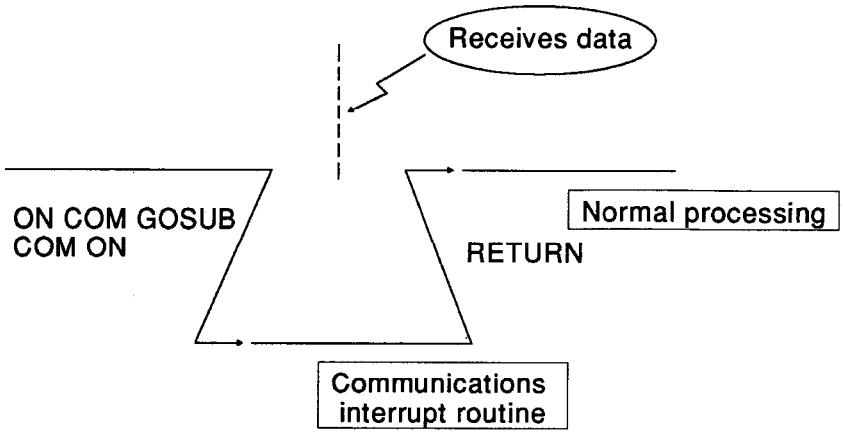
3) If data is not received normally, an error occurs. In this case, check the status of the communications line, the interface communications parameters, and the contents of the control table.

POINT

During communications with an external device, the RS/ER signals can be turned ON and OFF using the ZCNTL command.

7.4 Interrupt Processing from an External Device

In a BASIC program, the "communications interrupt routine" can be used to process data received from an external device.
Define the communications interrupt routine using the ON COM GOSUB command.
A line number or label should be specified along with the GOSUB command.
When data is received from an external device, the program jumps from the specified line or label to the interrupt routine.
To execute the communications interrupt routine, enable the communications interrupt.



In a communications interrupt routine, the BASIC commands can be used in the same way as in a BASIC program.
When processing is completed, the RETURN command can be used to return to the specified line or label.
To move the operation to a different line or label than the GOSUB line after completing the subroutine, input the line number or label after the RETURN command.

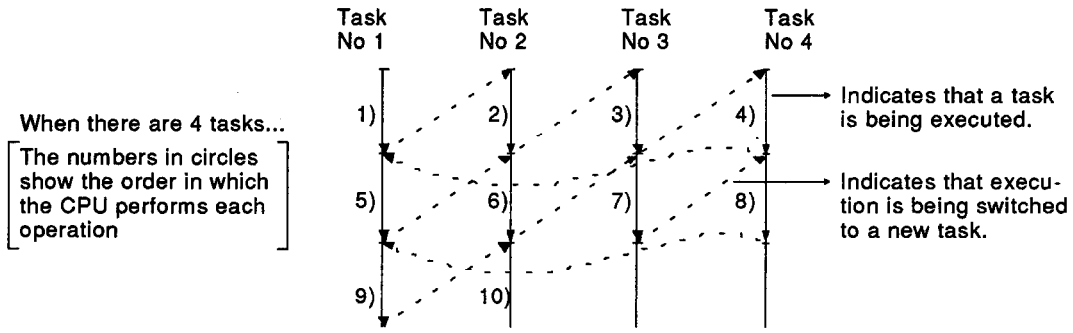
POINT

When the computer returns from communications processing to normal processing, another communications interrupt routine cannot be executed.

8. MULTITASKING

8.1 Multitasking

Multitasking refers to the ability of one machine to execute several programs simultaneously. AD51H-BASIC can execute up to 8 programs simultaneously on the AD51H using multitasking.
The term "multitask" consists of the words "task," which means "program," and "multi," which means "more than one."
The AD51H has one main CPU (central processing unit). By alternating the execution of each program, multitasking effectively makes execution simultaneous. Since CPU operation is very fast, to human eyes, the programs appear to be running simultaneously.



Each program is executed for a certain length of time specified by the operating system (50ms or 100ms), and then the operating system switches execution to the next program.
During parallel operation, the operating system (OS) alternates program execution (OS scheduling) based on each program's priority (for more information on the order of preference for execution, see section 8.4). Execution is also alternated whenever any of the following conditions occur:

- If a command is executed that performs input or output of data to or from the screen, keyboard, disk, peripheral device, external device, etc.
- If program execution is redirected (diverted, aborted, completed), or if a multitasking control command (execution synchronize, execution start or priority changing) is executed.
- If a program execution pause condition is canceled (if input is completed, if a specified time period is exceeded, etc.).
- When execution has continued for the predefined cycle time (50ms or 100ms in case of multi-statement execution).

For information on the commands which cause program execution to be rotated see appendix .

REMARKS

(1) The execution time period (cycle time) is switched between 50ms and 100ms with the AD51H system dip switch. For more details, see the AD51H/AD51H-S3 Users' Manual (Hardware volume).

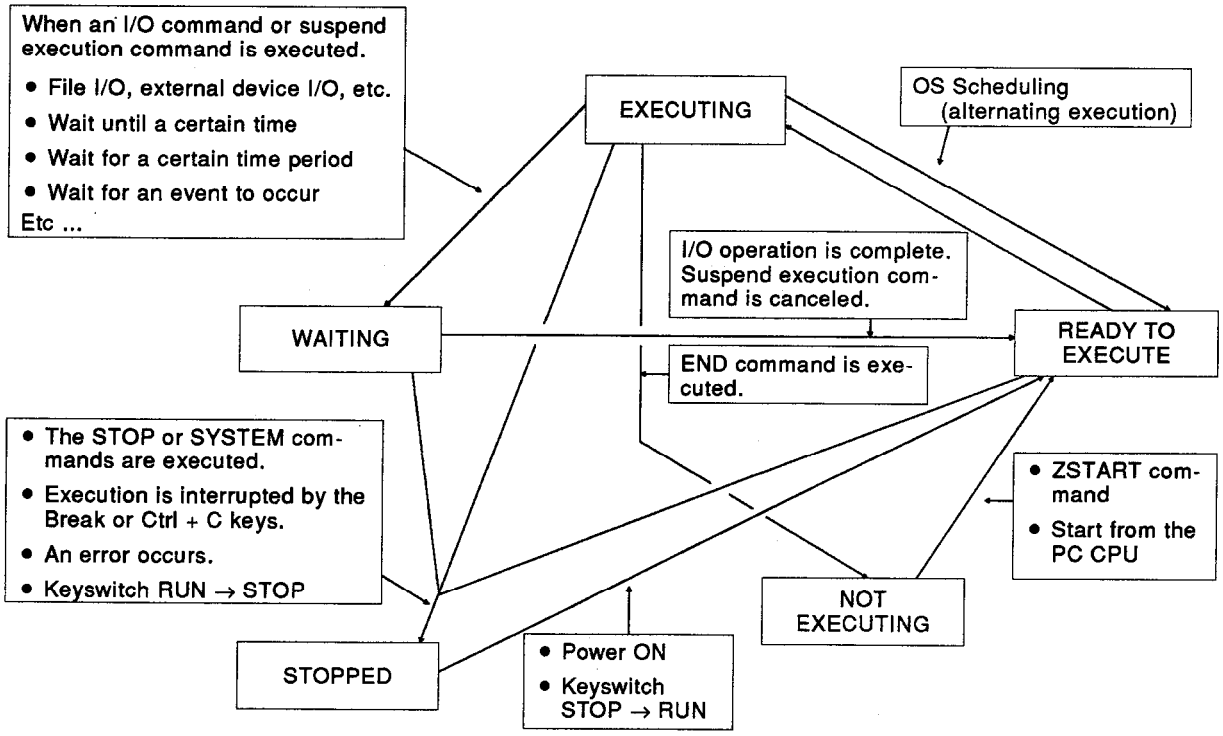
(2) Sometimes the task is switched at the end of the specified time period, but a command is in the middle of being executed. This presents no problems, operation proceeds normally.

(1) Changes in the Condition of BASIC Programs

This section describes the changes in the condition of a BASIC program when program execution is started.

The program's operating condition can be divided into four categories. Once execution has started, when a program is running together with other programs, the program's condition alternates between 1) to 3) below. Following execution of an END command, condition 4) is valid. If the SYSTEM command is executed, condition 5 becomes active.

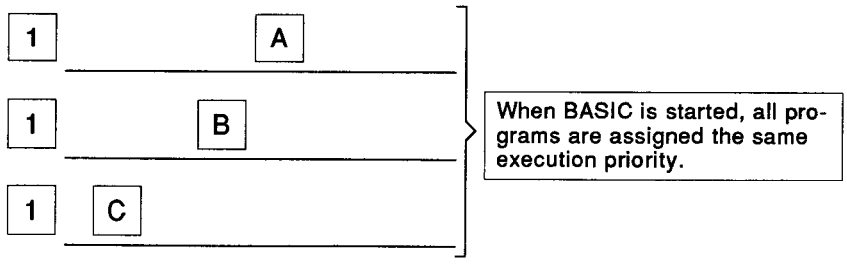
1) Executing	The program is currently executing.
2) Ready to Execute	When execution is switched back to this program and the OS gives the program permission to execute, this program is able to resume execution (execution condition will switch back to (1) "executing").
3) Waiting	When execution is suspended by a command which requires input from or output to the screen, keyboard, disk, peripheral device or external device, or by execution of a suspend execution command, the program remains in this condition until the I/O operation is complete or until the suspend execution command is canceled.
4) Not Executing	The program currently is not executing (excluding conditions 1) to 3), above). A "not executing" program can be restarted with the ZSTART command.
5) Stopped	This condition is active when none of the programs on the AD51H system are executing.



(2) Program Priority Level (Task Priority)

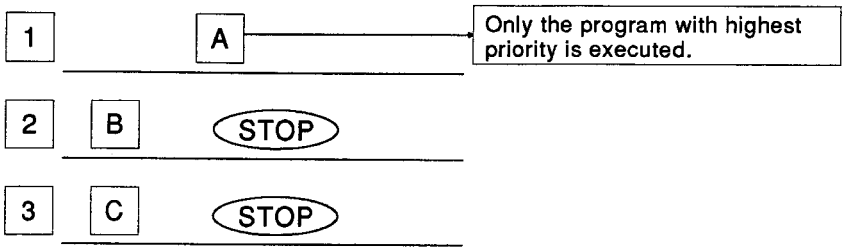
It is possible to assign an execution priority to each program executing under multitasking. When BASIC is started, all programs are assigned the same execution priority, so the main CPU executes all programs impartially under OS scheduling.

[Level of Priority]



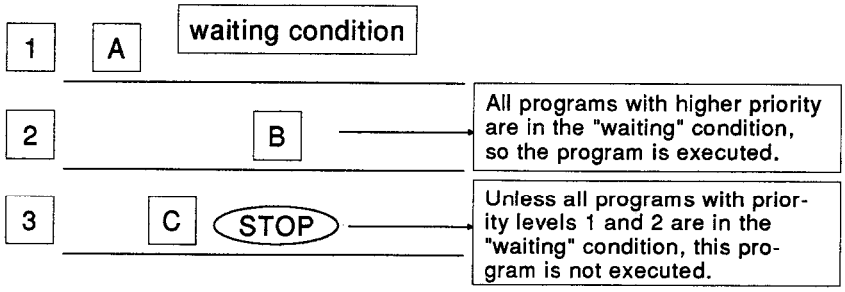
A program's priority level can be changed with the ZURGENCY command. When this happens, as long as no execution switching commands are executed (see Appendix), OS scheduling is not performed, and only the program with highest priority is executed.

[Level of Priority]



Except when execution switching commands are executed (see Appendix), programs with low priority do not run unless all programs with higher priority are in the "waiting" condition.

[Level of Priority]

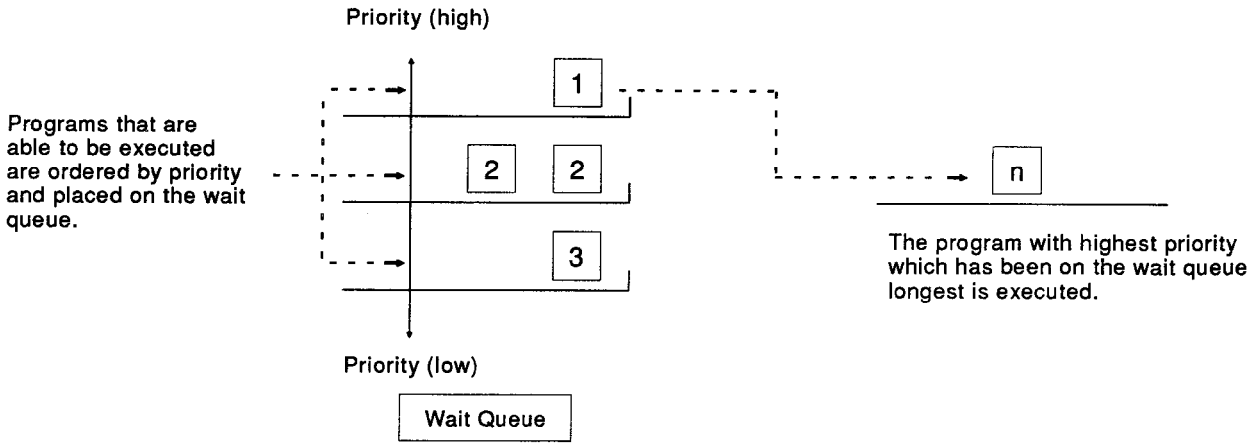


It is suggested that the priority level be increased only in extremely urgent cases, and that at all other times, programs should be executed impartially.

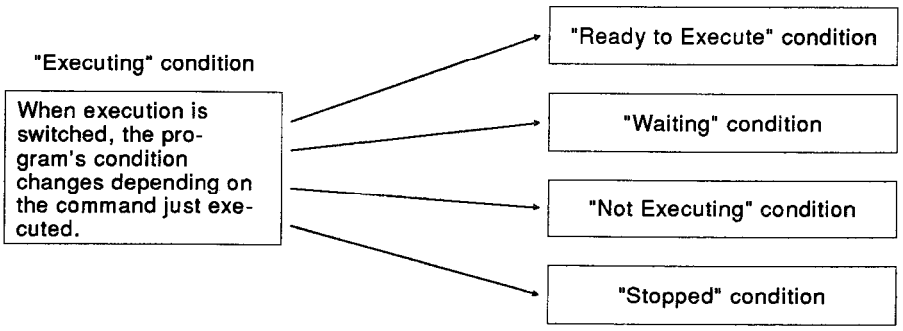
(3) Order of Execution when BASIC Programs are Executed Simultaneously

This section explains the order of execution of each program when multiple programs are executed, depending on the functional setup of the microcomputer in use.
In actuality, it is not only the BASIC programs that are running (executing). Depending on AD51H system scheduling, system programs and BASIC programs run together.

If several BASIC programs are running at once, when the AD51H alternates program execution, it chooses a program from the wait queue to be executed next. The wait queue contains all programs that are in the "waiting" condition (see the previous page). The AD51H assigns execution permission to the program with highest priority that has been waiting longest. The selected program is executed until program execution is alternated again.Therefore, simultaneous execution occurs according to this cyclic procedure.

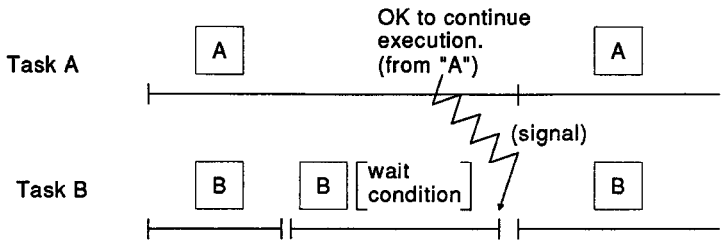


The program that is assigned execution permission by the OS as shown in the figure above assumes the "executing" condition. It executes a command, and then execution is alternated and the program assumes its next condition.



8.2 Synchronizing Execution (Event Processing)

Under multitasking, each program proceeds at its own pace. However, when it is necessary to send or receive data between programs, depending on each program's progress, timing may make correct data transfer impossible.
Event processing makes it possible for programs to be synchronized by forcing a program into "signal wait" condition (WAIT condition), and continuing execution only after a certain signal (event) is issued.



The signal that cancels a program's wait condition can be issued by another BASIC program or by a pre-defined AD51H internal device (EM).

In AD51H, the definition of events occurs in a determined order. Event processing can be used to synchronize execution of BASIC programs based on the events that are enabled.

(1) Conditions for Using Event Processing

Please keep the following points in mind when using event processing.

(a) If event processing predefines a certain event for any BASIC program, and the program contains a setting to enable release of the event, the event may be used. This predefinition/setting can be performed on any program. Moreover, the program in which an event is to be predefined and set does not even have to currently exist in memory.

(b) When an event is defined, event is release disabled. This is the same condition as when a setting (ZEVENT DISABLE) is used to disable event release.
In order to issue an event, it is necessary to enable the event by including a setting (ZEVENT ENABLE) in the program for which the event is defined.

(c) To synchronize execution with another BASIC program using event processing, the program that waits for the event must be in the "waiting for event" condition before the event is issued.
When an event is issued, if there are no programs waiting for the event, that event's release becomes invalid (ignored).

(d) If event processing is in use, when the AD51H system is stopped or set up, it is necessary to disable all events to be handled. (This initializes event information managed by the system.)
Either of the following methods is possible:

1) Turn the power off and on again, or use the RESET switch.

2) Re-define all handled events either in the first BASIC program that is executed during AD51H system setup, or in the last BASIC program to finish execution. (Re-definition is performed with the DEF ZEVENT command.)

Example

```
FOR 1%=0 TO 16
DEF ZEVENT 1%
NEXT 1%
```

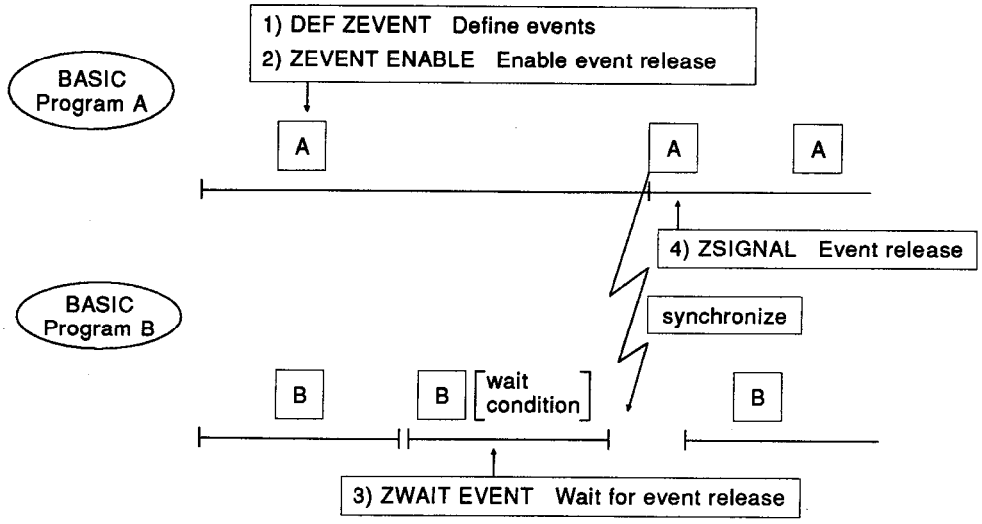
REMARK

The AD51H system cannot perform event processing based on PC CPU bit devices.
However, a stopped BASIC program can be started by a directive from the PC CPU. For details, see section 5.2.

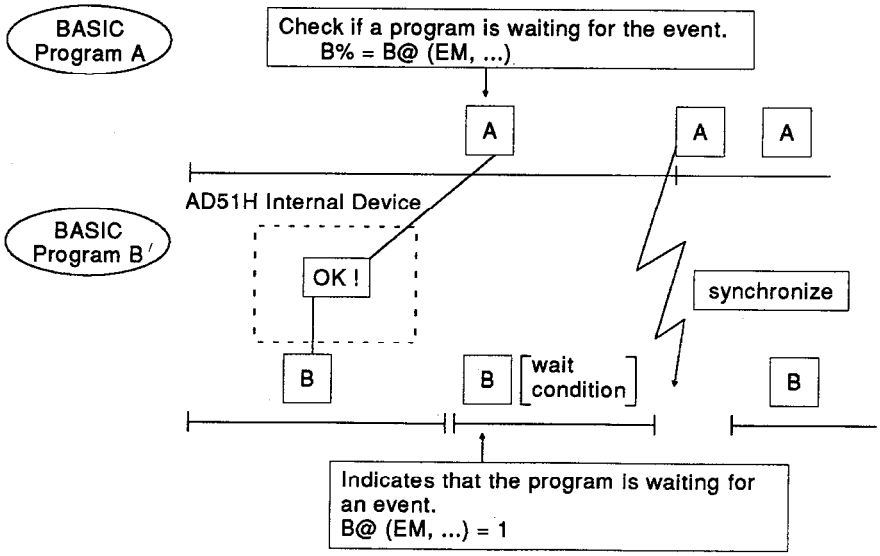
(2) Order of Command Execution under Event Processing

When execution of a BASIC program is controlled with event processing (execution start, execution restart), the order of command execution is as shown below. The commands should be executed in order starting with (1).

(a) Synchronizing execution within a BASIC program:



As shown above, once 1) and 2) have been executed in one of the BASIC programs, any other BASIC program can wait for the specified event, then the event can be issued. If 4) is executed before 3), execution is not synchronized.
In order to keep the released event from becoming disabled, use an interlock to synchronize execution based on an AD51H internal device.

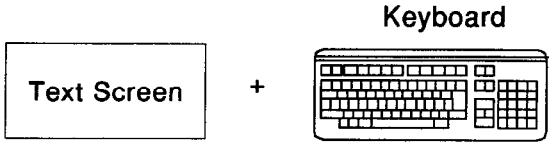


8.3 Using Devices (Resources) under Multitasking

The term "resources" refers to all hardware and software that can be used by the program. These include the display screen, printer, files, disks, option boards, memory, etc. When these are used by several executing programs, several of the various devices can be grouped together and considered one resource, or one device can be divided and treated as several resources.

Example

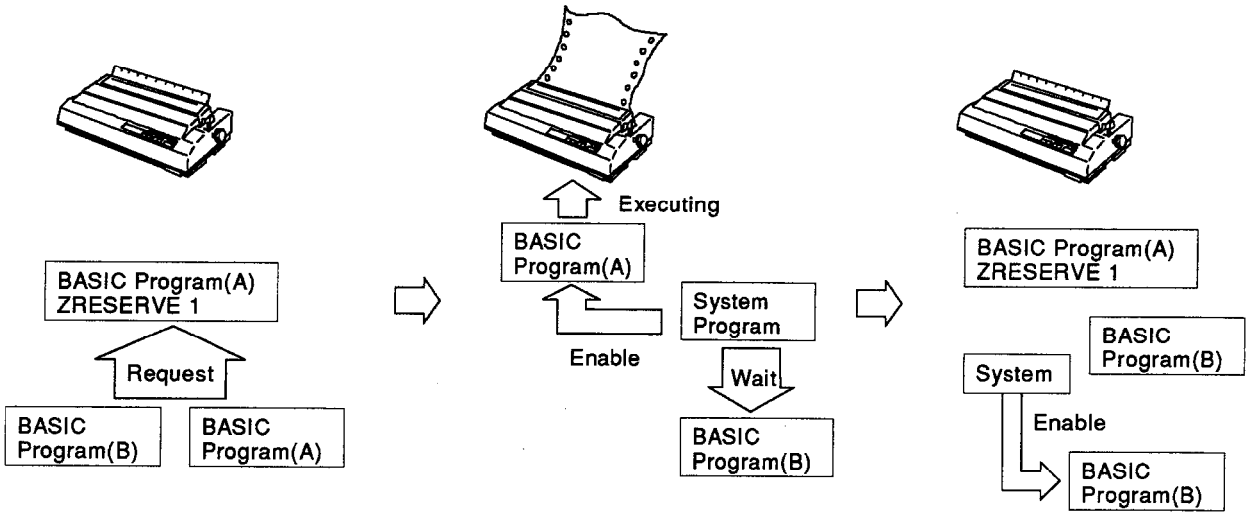
Grouping several resources together as one resource:



Dividing one resource into separate resources:
(Extended relay EM0 to EM1023, an AD51H internal device)

Exclusive access management for resources ensures that when several programs are executing simultaneously, and one program is using a particular resource, no other programs can use that resource. In this section, devices are used assuming that exclusive access management is controlling access to all resources by several simultaneously executing programs.

- Exclusive Access Control - Printer Example



ZRESERVE 1
Two programs request to use the same printer.

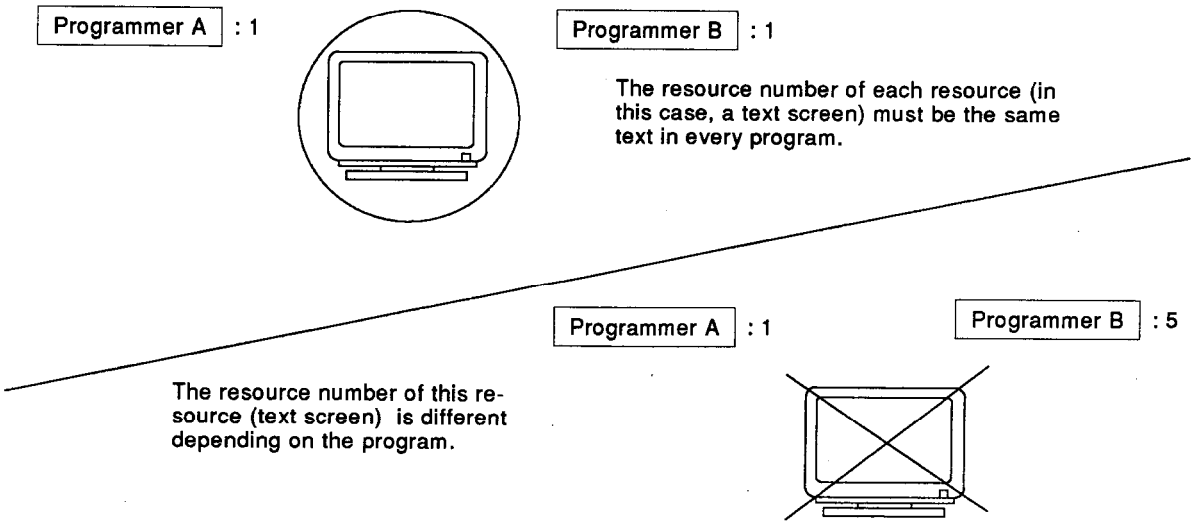
The program that requested to use the printer first is given permission.
The other program must wait.

Two programs request to use the same printer.

(1) Conditions for Resource Access Management

Please keep the following points in mind when exclusive access management is in force.

(a) Resource access management must be completely agreed upon by all programmers in the system. If there are discrepancies between programmers, resource access management has no meaning.



(b) A program that uses a resource that has an assigned resource number (see section (2)) must execute the ZRESERVE command immediately before using the resource. After the program is finished using the resource, it must execute the ZRELEASE command so that the resource may be used by other programs.

When a program executes the ZRESERVE command, but the specified resource is already being used by another program, the program that issued the ZRESERVE command is paused until the other program is finished using the resource.
If the program cannot wait a long time for the resource, when the ZRESERVE command is executed, a timeout period can be specified.
If the resource is not available within the prescribed time limit, an error occurs, and through error management, another operation can be begun.

(c) When using resources that have an assigned resource number, try as much as possible to perform access management one at a time.

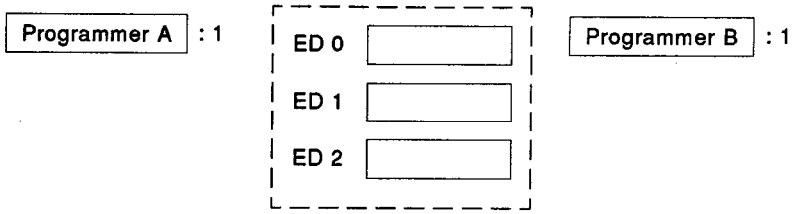
(d) When several numbered resources are used simultaneously, it is necessary to decide an order of resource use based on the resource numbers specified in each ZRESERVE command (program deadlock prevention). For example, proceed from smaller resource numbers to larger ones or vice-versa.

(2) Assigning Resources to Resource Numbers

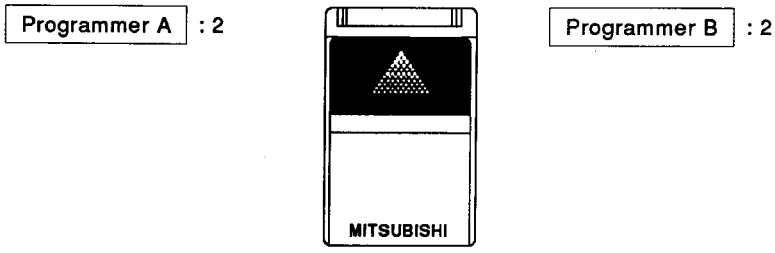
This section explains how to assign a resource such as the screen or printer to a resource number to be used by the ZRESERVE and ZRELEASE commands for resource access management.

To assign resources to resource numbers, before the programs are written, all programmers must agree upon what numbers to assign to what resources. There is no command to assign a resource to a resource number, the assignment is carried out by the mutual agreement of all programmers to use the same numbers to specify each resource. For example:

- When reading or writing data to or from ED0 to ED2, the ZRESERVE command is executed using resource number 1.



- When reading or writing to or from file "ABC,NEW" on drive 0, the ZRESERVE command is executed using resource number 2.



File "ABC,NEW"

The above resource numbers are decided upon beforehand and used consistently throughout every program.

The following is an example of how to assign resources to resource numbers.

(a) All resources that are used by simultaneously executing programs should be considered.

(b) Of the resources in (a), those that are used simultaneously by several programs should be selected.

(c) Of those selected in (b), resources that will cause an error if they are used simultaneously by several programs should be selected.

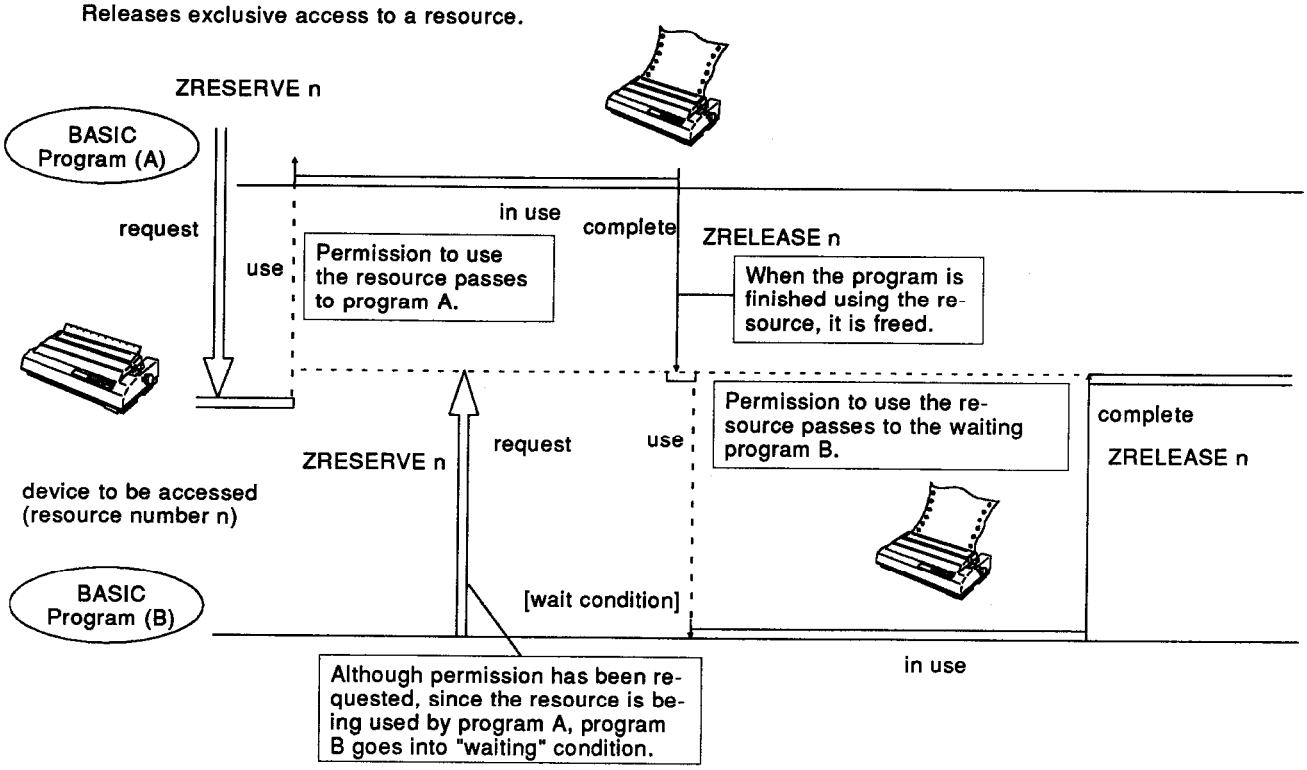
(d) Of the resources selected in (c), multiple resources that are used simultaneously should be made into one resource and single resources should be divided into separate resources. The resources to be assigned resource numbers should then be clarified.

(e) The resources clarified in (d) should be assigned a resource number between 0 and 64 (mutual decision).

(3) Order of Command Execution under Resource (Exclusive) Access Management

This section explains the order in which commands are executed when resource exclusive access management is active.

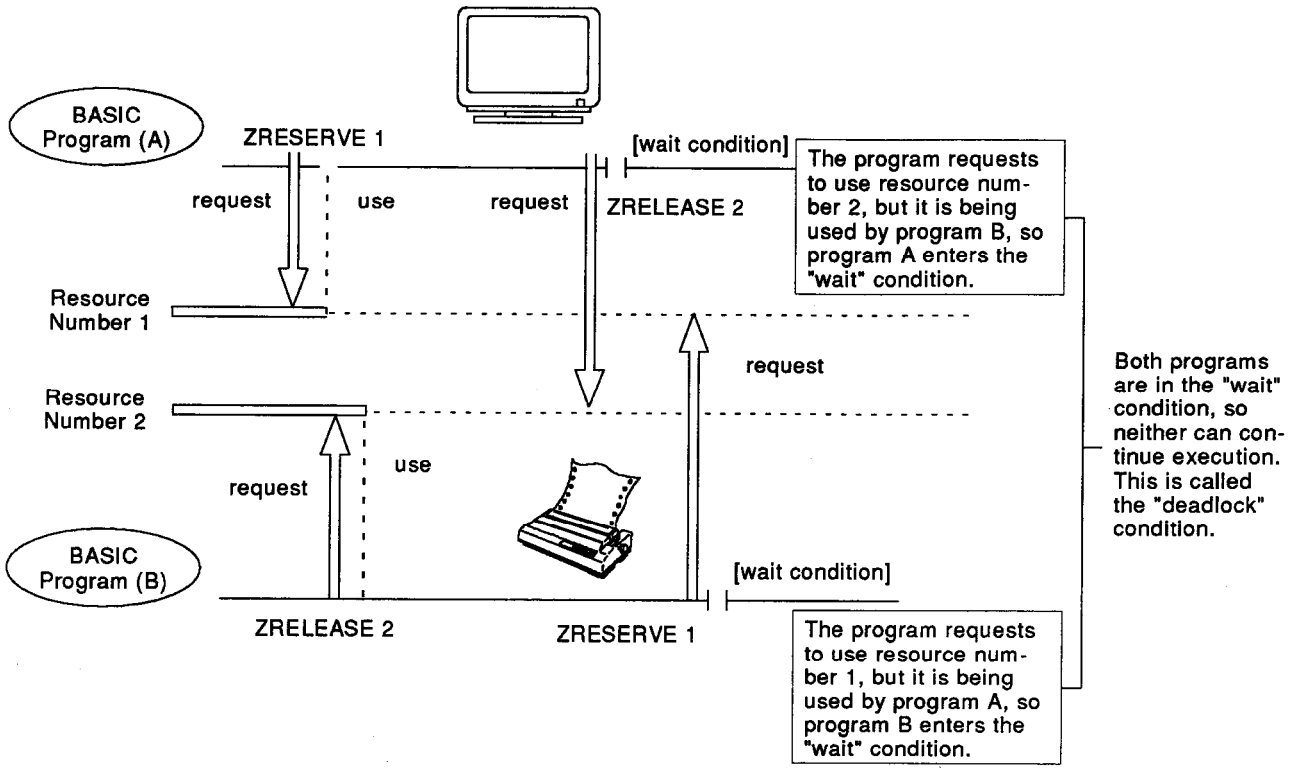
ZRESERVE:
Requests exclusive access to a resource.
ZRELEASE:
Releases exclusive access to a resource.



As shown above, to use a resource that is assigned resource number n, the ZRESERVE command must be executed before the corresponding resource can be used. When finished using the resource, the ZRELEASE command must be executed.

However, if simultaneously executing programs wish to use several resources at once, it is necessary to proceed in an order based on the resource numbers in each ZRESERVE command in a way universally agreed upon. If this is not done, two programs may both be locked in the "wait" condition (deadlock).
An example of deadlock follows.

Deadlock Example



In this case, if a timeout period is specified in each ZRESERVE command, when the time period expires, a timeout error occurs, allowing the execution pause condition to be canceled.
However, since a timeout error is not considered a good way to manage deadlocks, it is suggested that the programs be edited.

(4) Types of Resources that are Managed by the AD51H Operating System or by Each Program

The following resources do not require exclusive access management.

(a) Resources managed by the AD51H operating system:

- [Channel numbers] specified for data transfer with external devices
- [File numbers] specified for I/O with a file or device
- [Priority numbers] which decide program execution order
  However, sometimes a high number is used exclusively. For example, in an unusually urgent situation, to make one program have the highest priority.
- [Character strings] assigned to function keys

(b) Resources managed within each program individually:

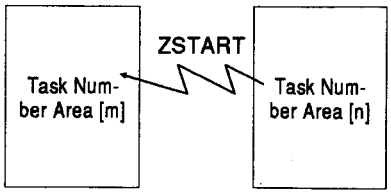
- [Variables] and [arrays] used in the program

All resources except for the above can be shared by all programs. For example:

Text Screen
Keyboard
Files, Disks
AD51H Internal Devices
Event Numbers
Resource Numbers
etc.

8.4 Starting Another Program from within a Program

When running under multitasking, it is possible to start a new program in any desired task number area using the ZSTART command.



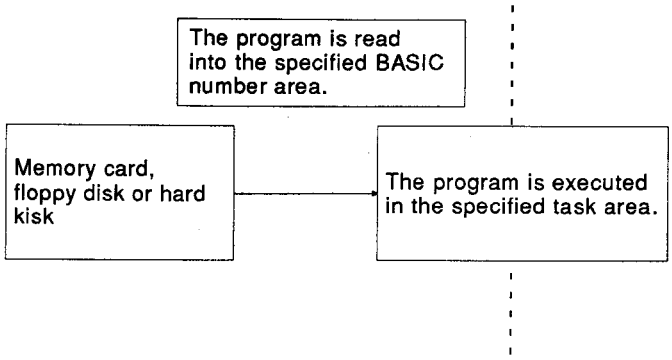
POINT

When starting a program with the ZSTART command, the AD51H system mode SET command must be used to set the corresponding task number area to 'START', 'BOOT', 'IT', or 'ON'. If it is set to 'OFF', the ZSTART command cannot be used to start a program.

There are two ways to use the ZSTART command to start a program.

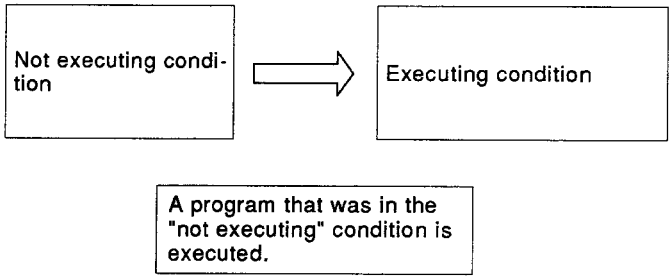
1) Read the program from a memory card into the specified task number area and start the program.

   ZSTART <task number area>, <program to read>



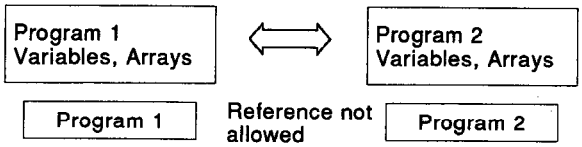
2) Start a program that is currently residing in a task number area, but is in the "not executing" condition due to encountering an END command.

   ZSTART <Task number area>



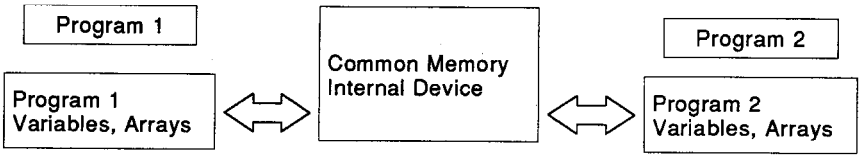
8.5 Transferring Data between Programs

Under multitasking, the contents of variables and arrays can only be referenced within the program in which they are used. To transfer data between programs, a special procedure must be carried out.

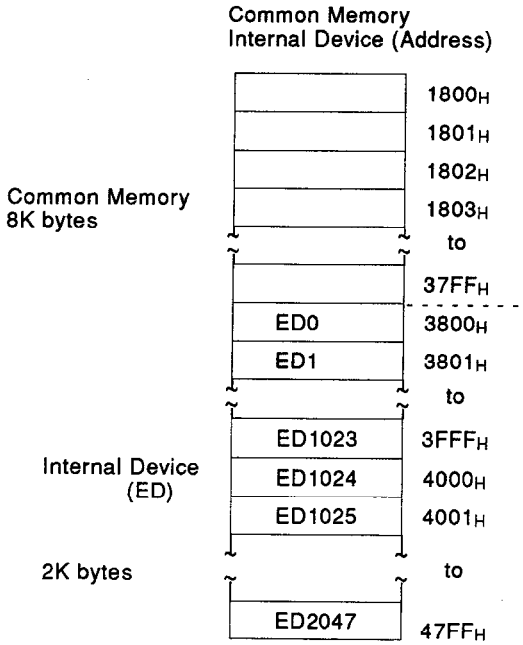


8.5.1 Common Memory and Internal Devices

Common memory and internal devices (ED) are used as memory which every task can access in the same way. By using this area, data can be exchanged between tasks.



The PUTMEM and GETMEM commands are used to access common memory and the internal devices.
Memory is composed as shown below:

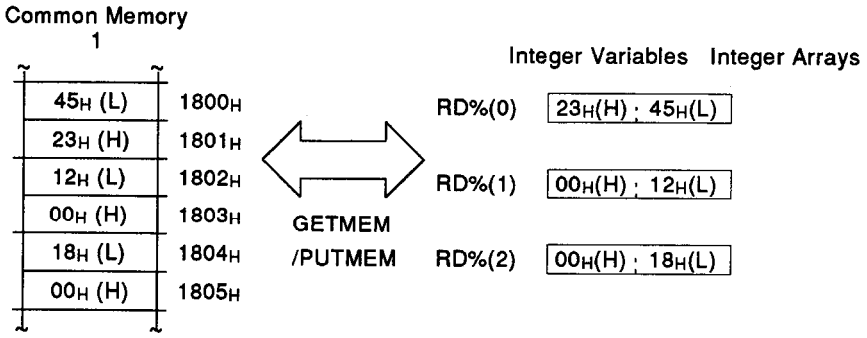


POINT

The PUTMEM and GETMEM commands are used to access both common memory and buffer memory. If an address in the range 0H to 17FFH is specified, buffer memory will be affected. Please be careful when using these commands.

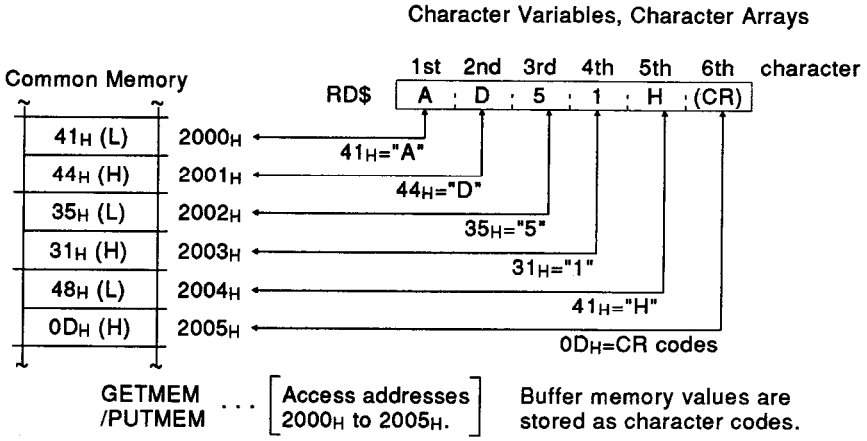
Data is stored as follows:

- Numeric Data

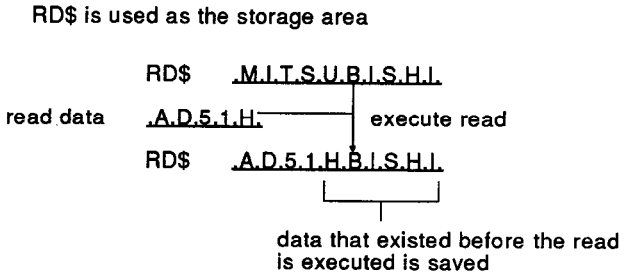


(Access addresses 1800H to 1805H.)

- Character Data



- When using the GETMEM command to read out data, if the number of bytes (one byte = one character) of data currently stored in the variable which is to receive the data is greater than the number of bytes of data to be read in, the data is stored as shown below.



REMARK

Internal device ED can also be read from and written to using special variable W@ (ED, n). See section for details 2.9.3.

8.5.2 The Message Port

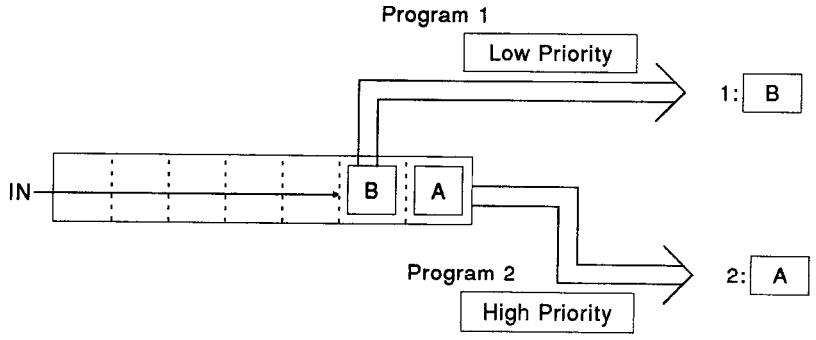
By creating (defining) a place in memory to send and receive messages (message port), AD51H BASIC programs can send and receive messages from each other just like sending letters through the mail. The message port can also be used to synchronize program execution.
When the message port is used to synchronize program execution, it is not necessary to think about "wait" condition interlock as with event processing. The "wait" condition is automatically created.

(1) Conditions for Using the Message Port

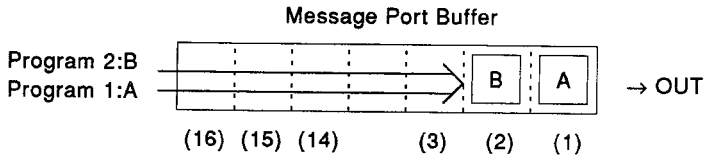
Please keep the following points in mind when using the message port:

- The message port must first be opened by one of the BASIC programs before it can be used. It does not matter which program opens the message port. Also, it is not necessary for the program which opened the message port to remain in memory.
- The length of one message must be less than or equal to 256 bytes.
- Up to 32 message ports can be opened in the AD51H system. To distinguish between the various message ports, a message port number between 0 and 31 is assigned to each port.
- Each message port can store up to 16 messages asynchronously. Messages can be read from the message port in the order in which they were received (First In First Out (FIFO) style), or in order of program priority (Priority style). The style in which data is read from the message port is decided for each message port number separately.

Priority Style ..... Message port numbers 0 to 15

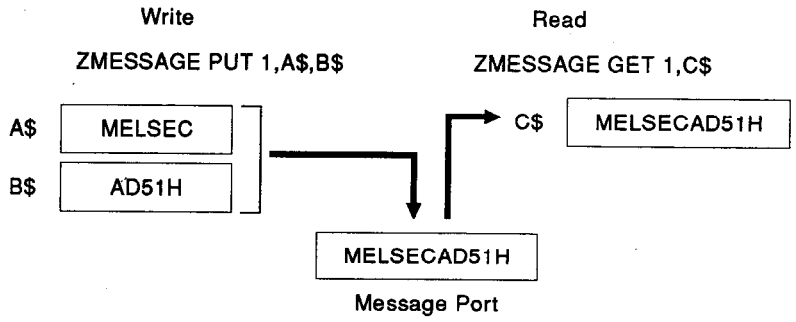


FIFO Style ..... Message port numbers 16 to 31



- If the total number of messages stored in all message ports exceeds 64, an error occurs. However, when a message is read, the number of messages stored decreases by one.
- When reading a message from a message port, if there are no messages in the specified message port, the program enters the "wait" condition until a message arrives at the port.

- When using the message port, the following points must be decided beforehand.
  1) The length of messages to be input to and output from the port (the number of bytes)
  2) The order format (style) of the message port (FIFO style or Priority style)
- When multiple data items are used as a message, even if the message is written as shown below, it is handled as a single message when it is read.



This is because the message port does not support separation within a message.
In order to partition data within a message, a special procedure must be written.

(2) How to Use the Message Port

The following points about message port data and methods of I/O should be noted.

1) The message port is defined:

ZMESSAGE <message port number> LEN = <message length>

When a message port is defined, BASIC creates it in main memory.

The message port can be defined with the ZMESSAGE command by <u>any of the programs</u> running under multitasking.

2) The message port is opened:

ZMESSAGE OPEN <message port number>

The message port must be <u>opened</u> with the ZMESSAGE OPEN command by <u>each of the programs</u> running under multitasking.

3) A message is written to the message port:

ZMESSAGE PUT <message port number>, <data to send>, ...

3)' A message is read from the message port:

ZMESSAGE GET <message port number>, <variable to store message in>, . . .Δ <timeout period>

The parameter <timeout period> is explained in section (3) part 2).

4) When finished with message I/O, the message port is closed:

ZMESSAGE CLOSE <message port number>

A message port that is closed can be reopened with the ZMESSAGE OPEN command.

5) When finished using a message port, the message port is deleted:

ZMESSAGE KILL <message port number>

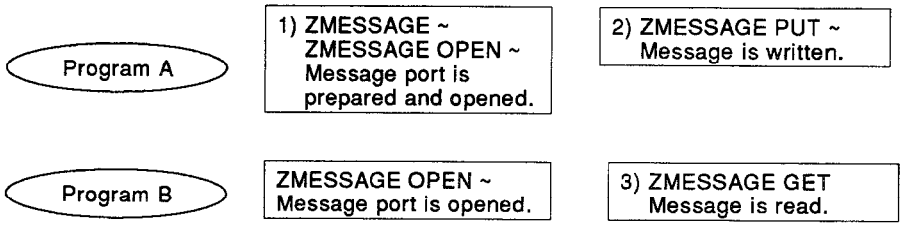
When a message port is deleted, it is erased from main memory. In order to use it again, it must be redefined.

The message port can be deleted with the ZMESSAGE KILL command by <u>any of the programs</u> running under multitasking.

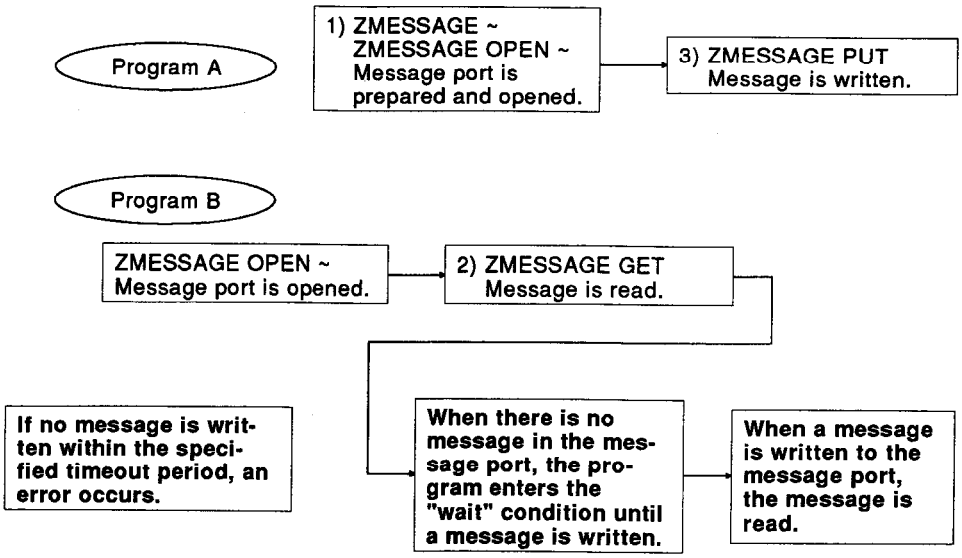
(3) Order of Command Execution when Using the Message Port

This section explains the order in which commands are executed when the message port is used to send and receive messages or to synchronize program execution.

1) The figure below shows what happens when program A sends a message, and then program B reads the message.



2) The following illustrates how program execution is synchronized if program B tries to read a message before program A has written it.



Unlike event processing, when the message port is used, the "wait" condition is automatically generated, so it is not necessary to create an interlock within the program.

REMARK

The program that writes a message to the message port does not control the transfer of the message.
To manage a message based on the program that wrote it, write the program name as part of the message, and then the receiving program can use the program name as a factor in message management.

9. ERROR MANAGEMENT

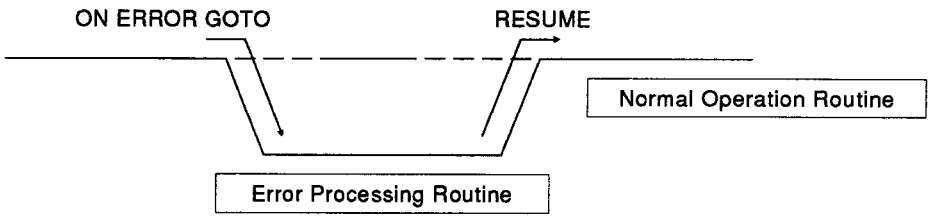
If an error occurs while a BASIC program is executing, an error message is displayed and program execution is stopped.

A program should be written so that errors with various causes will not occur. However, sometimes an error will occur as a result of something which cannot be anticipated.

Because of this, an "error processing routine" can be created in BASIC so that even if an error occurs, no error message is displayed and execution is not stopped.

9.1 Defining Error Processing

The procedure to be performed when an error occurs, called the "error processing routine" is defined using the ON ERROR GOTO command. If an error occurs, execution is begun starting from the line number or label specified in the ON ERROR GOTO command.



Within the error processing routine, BASIC commands can be used to correct the cause of the error.

When error processing is complete, the RESUME command is used to return execution to the normal operation routine.

POINT

When error processing is complete, the RESUME command must be used to return to the normal operation. If the GOTO command is used, the next time an error occurs, error processing cannot be performed.

REMARK

To deactivate an error processing routine that is already defined, use ON ERROR GOTO 0. After executing this command, if an error occurs, an error message is displayed and the program is stopped.

9.2 Error Types and How to Identify the Error Location

The following two functions can be used within the error processing routine to determine the error type and the line number where the error occurred.

- ERR function ..... Returns the error type of the error that occurred.

POINT

The ERR function is used to determine what type of error occurred.

- ERL function ..... Returns the line number where the error occurred.

POINT

The ERL function is used to determine where in the program the error occurred.

The ERR function returns the error type as an error code. For information on error types and their corresponding error codes, see the error code table in appendix .

By combining the ERR and ERL functions, the error which occurred can be determined in detail. These functions are used in the error processing routine so that depending on the error that occurred, a correct procedure can be chosen and performed.

Example

```
  5 ON ERROR GOTO 100
 10 INPUT"X=";X
 20 INPUT"Y=";Y
 30 A=EXP(X):PRINT"EXP(X)=";A
 40 B=10^Y : RRINT"10^=";B
 50 END
100 IF ERR=6 AND ERL=30 THEN ——— An overflow error occurred in line 30.
       PRINT "Please select a lower value for X."
110 IF ERR=6 AND ERL=40 THEN ——— An overflow error occurred in line 40.
       PRINT "Please select a lower value for Y."
120 RESUME 10
RUN
X=?100
Y=?2
Please select a lower value for X.
X=?2
Y=?2
EXP (X)  =7.38906
10^Y=100
OK
■
```

9.3 Error Processing Difficulties

The following points should be kept in mind when creating an error processing routine.

(1) Error Prevention within the Program

The error processing routine should be considered a "last attempt" to prevent program execution from being stopped by an error. Whenever possible, the program should be written so that errors can be prevented without resorting to the error processing routine.

Example

```
10 INPUT "A=";A
20 INPUT "B="'B
30 C=A/B
40 PRINT "A/B=";C
```

In this program, if B is set to 0, in line 30 a "Division by zero" error will occur.

```
10 INPUT "A=";A
20 INPUT "B=";B
25 IF B=0 GOTO 20
30 C=A /B
40 PRINT "A=";C
```

By adding line 25, if B = 0, line 30 will not be executed.

(2) Measures for Avoiding Errors that Cannot be Processed

Within the error processing routine, the ERR and ERL functions must be used together to decide whether the error which occurred can be corrected using the error processing routine. If this is not done, and incorrect error management is performed, the operation of the system may become anomalous.
If an error occurs that cannot be corrected, it is suggested that the ON ERROR GOTO 0 command be used to abort the error processing routine and halt program execution.

Example

Error processing occurs in the case of a memory card insertion error. For any other error, program execution is halted.

```
  10 ON ERROR GOTO 1000
     :
 100 OPEN"0:DATA\110.DAT"FOR INPUT AS #1
     :
 200 OPEN"0:DATA\120.DAT"FOR INPUT AS #2
     :
 900 END
1000 IF ERR<>53 THEN ON ERROR GOTO 0 ..... Check if the error is a "File not found" error. If it is not,
                                            program execution is halted.
1010 IF ERL=100 THEN PRINT "Please insert the
     memory card that contains 110.DAT" :GOTO 1040 ....
                                                       :..... Check whether the error occurred in line 100 or 200.
1020 IF ERL=200 THEN PRINT "Please insert the
     memory card that contains 120.DAT" :GOTO 1040 .....:
                                                          [If the error occurred in line 100 or 200, the corresponding error management procedure is performed.]
1030 ON ERROR GOTO 0 ..... If the error did not occur at line 100 or 200, program execution is halted.
1040 PRINT "Press any key when ready..."
1050 K$=INPUT$(1)
1060 RESUME ..... Return and restart execution at the line where the error occurred.
```

10. DEBUGGING A PROGRAM

When a program does not execute correctly, to fix the program so that it executes properly is called "debugging."
This section explains procedures for debugging a program, and the debugging commands that can be used.

10.1 Debugging Programs that Execute Simultaneously under Multitasking

When several programs are executing simultaneously (parallel operation under multitasking), the following general procedures apply to debugging programs.

(1) Individual Debugging (unit testing) ..... Execute each program one by one, setting many different conditions to change the flow of execution, and verify that the programs execute properly.

(2) Combined Debugging (group testing) ..... Several programs which together perform one operation should be grouped together. Once all programs have been grouped, execute each group separately and verify that all programs execute properly.

(3) Total Debugging (system testing) ..... All programs are executed to verify that each program executes properly.
When total debugging (system testing) is performed, if all programs execute properly, debugging is complete.

10.2 Commands Used for Debugging a Program

This section describes the commands typically used to debug a program.
For details on each command, see chapter 11.

(1) TRON·TROFF Commands

The TRON command is used to follow the flow of execution. After the TRON command is executed, the line number of every line that is executed is displayed in brackets. The operation of IF~GOTO~branches, or of FOR~NEXT loops can be clearly understood.
To stop the display of line numbers, execute the TROFF command.

```
10  C=10
20  FOR I=1 TO 3
30   C=C+1
40  NEXT I
50  TRON
60  FOR I=1 TO 3
70   C=C+1
80  PRINT I;C
90  NEXT I
100  TROFF
110  C=C+1
120  PRINT C
130  END
```

```
RUN
[60][70][80] 1   14
[90][70][80] 2   15
[90][70][80] 3   16
[90][100] 17
OK
```

(2) The STOP and CONT Commands [Break] ([CTRL]+[C]) Key

When the stop command is executed, program execution is temporarily suspended (aborted).
Also, if [Break] ([CTRL]+[C]) key is pressed during program execution, the program is temporarily halted (aborted) just as when the STOP command is executed.
The CONT command is used to restart execution of a stopped program (from the command following the break).

```
10  C=1
20  FOR I%=0 TO 3
30   PRINT I%
40  NEXT I%
50  PRINT "Number of times = ";C
60  STOP
70  C=C+1
80  GOTO 20
```

```
RUN
 0
 1
 2
 3
Number of times = 1
Break in 60
OK
CONT
 0
 1
 2
 3
Number of times = 2
Break in 60
OK
```

POINT

If an edit, append or delete operation is performed on a stopped program, it cannot be restarted with the CONT command.

(3) The PRINT Command and the LET Command

This section explains how to examine the values of the variables and arrays used by a program that has been stopped using the method described in part (2), above.
The PRINT command can be used to display the current value of variables on the screen.
The LET command can be used to change the current value of variables.

```
10  DIM IN$(2)
20  FOR I%=0 TO 2
30   INPUT IN$(I%)
40  NEXT I%
50  STOP
60  FOR I%=0 TO 2
70   PRINT IN$(I%)
80  NEXT I%
90  END
```

```
RUN
? MITUBISHI
? MELSEC
? AD51H
Break in 50
OK
PRINT IN$(0)
MITUBISHI
OK
IN$(0) = "Mitsubishi"
OK
PRINT IN$(2)
AD51H
OK
IN$(2)="BASIC"
OK
CONT
Mitsubishi
MELSEC
BASIC
OK
```

If the commands shown below are used in the BASIC program, debugging becomes simpler.

- ON ERROR GOTO·RESUME commands

  Used to prevent program execution from being aborted when an error occurs.

- ERROR command

  Causes any desired error to occur in order to see if the flow of program execution is correctly altered.

- ERR·ERL functions

  Verifies the error code or line number of the error that just occurred.

11. COMMANDS AND FUNCTIONS

This chapter explains the general commands used in BASIC and the functions that can be used within commands.

Each entry is arranged in the following format:

The name of the command or function

The purpose of the command or function

Indicates whether it is a command or a function.

Shows the origin of the command name.

FORMAT command — **MELSEC-A**

FORMAT

- Initializes the file area of a memory card (performs a logical format).

The syntax for entering the command or function

SYNTAX	**FORMATΔ <drive number>**	
	drive number	Specifies the memory card to be initialized (logically formatted).
EXAMPLE	**FORMAT 0**	The memory card inserted in memory card slot 1 is initialized (logically formatted).

Parameters used in the command or function syntax

The meanings of different arguments are explained.

An example of using the command or function

Description or important points

DESCRIPTION

- The **FORMAT** command initializes (logically formats) the file area of a memory card so that BASIC program and data files can be written to it.
- If the file area of a memory card has not been initialized (logically formatted) with the **FORMAT** command, BASIC program and data files cannot be written to the memory card
- Simply initializing (logically formatting) a memory card with the BASIC **FORMAT** command does not make the memory card usable. To make a memory card usable, it must first be completely initialized (physically formatted) in file maintenance mode or in system mode. Following the complete initialization (physical format) the BASIC **FORMAT** command is executed to initialize (logically format) the file area.

MANUALS FOR REFFERENCE

Complete initialization (phycical format) of a memory card in file maintenance ..... SWOIX-AD51HP: AD51H Software Package Operating Manual

Complete initialization (phycical format) or a memory card in system mode. ..... AD51H Users' Manual

- The **<file number>** specifies which memory card's file area is to be initialized (logically formatted). The following shows the valid settings:

Initialize the file area of the card inserted in Memory Card Slot 1 0

Initialize the file area of tha card inserted in Memory Card Slot 2 1

Important points to remember when using the command or function

POINT

When the file area of a memory card is initialized with the FORMAT command, all program and data files stored on the memory card are lost. There is no way to retreive lost data. Before using the FORMAT command, be sure to thoroughly verify the contents of the memory card.

An example of the using the command or function in a program

PROGRAMMING EXAMPLE

Additional explanation, or other commands or functions used in conjunction with this one

REMARK

Refer to the SW[ ]IX-AD51HPE Software Package Operating Manual, and the AD51H/AD51H-S3 User's Manual.

The symbols used in describing command and function syntax have the following meanings:

(1) Any alphabetic characters can be entered as is.
(2) Anything in square brackets is optional (can be omitted).
(3) Anything in parentheses or quotes should actually be entered in parentheses or quotes.
(4) Any item followed by an ellipsis (...) indicates that the particular item can be repeated many times within the limits of the line length.
(5) The symbol indicates that at least one whitespace (space) character is necessary.
(6) Anything enclosed in the symbols indicates an item to be specified by the user. Please set a parameter argument according to your objectives.

- Returns the absolute value of a numeric expression.

ABSolute

SYNTAX	**ABS (<numeric expression>)**	
	numeric expression	Specifies the value for which the absolute value is to be calculated.
EXAMPLE	**A=ABS(-3.14)**	The absolute value of -3.14, which is 3.14, is stored in A.
	B=ABS(0)	The absolute value of 0, which is 0, is stored in B.
	C=ABS(3.14)	The absolute value of 3.14, which is 3.14, is stored in C.

DESCRIPTION

- The **ABS** function returns as its value the absolute value of **<expression>**.
- The value returned is normally a positive number or zero.
- If a double precision real number is used as the **<expression>**, the value returned is double precision. If not, the value returned is single precision.

PROGRAMMING EXAMPLE

```
10 'Returns the absolute value
20 A=-2.56
30 B=0                                  :'Define numeric values
40 C=2.56
50 PRINT "ABS(A)=";ABS(A)
60 PRINT "ABS(B)=";ABS(B)               :'Display the absolute values
70 PRINT "ABS(C)=";ABS(C)
80 END

RUN
ABS(A)= 2.56
ABS(B)= 0
ABS(C)= 2.56
OK
```

- Returns the character code of the first character of a character string argument.

ASCii

SYNTAX	**ASC (<character string expression>)**

	character string expression	Specifies the character for which the character code is to be calculated.
EXAMPLE	**A=ASC ("BASIC")**	The character code of the first character of the string ("B"),which is 42_H, is stored in A.
	C=ASC ("D")	The character code of character "D", which is 44_H, is stored in C.

DESCRIPTION

- The **ASC** function returns as its value the character code of the first character in **<character string argument>**.
- If the empty string is specified for **<character string expression>**, an **"Illegal function call"** error occurs.

PROGRAMMING EXAMPLE

```
10  :'Displays the character code of the first character of a character string
20  A$="AD51H-BASIC"                          :'Define the character string
30  CODE=ASC(A$)                              :'Get the character code
40  PRINT "Character=";CHR$(CODE)             :'Display the character
50  PRINT "Decimal=";CODE                     :'Display the character code in decimal
60  PRINT "Hexadecimal=";HEX$(CODE)           :'Display the character code in hexadecimal
70  END

RUN
Character=A
Decimal=65
Hexadecimal=41
OK
```

REMARK

See CHR$, HEX$.

For converting character codes to characters, see the CHR$ function.

- Returns the arctangent (tan-1) of a numeric expression.

Arc TaNgent

SYNTAX	**ATN (<numeric expression>)**

	numeric expression	Specifies the numeric value or numeric variable for which the arctangent is to be calculated.
EXAMPLE	**A=(180/3.141592)*ATN(1)**	The arctangent of 1 is calculated, and the value of the resolved expression is stored in A.

DESCRIPTION

- The **ATN** function returns the arctangent (**tan**$^{-1}$) of the value of **<numeric expression>**.
- The **<numeric expression>** can be specified as any desired type of number, but the **ATN** function normally calculates in single precision.
- The unit of the numerical value that is given is radian.

PROGRAMMING EXAMPLE

```
10  'Use the ATN function to calculate an angle measure
20  HI=1732/1000                          :'Define a numeric value
30  RAD=ATN(HI)                           :'Calculate the angle measure
40  DEG=(RAD*180)/3.141592653#            :'Convert radians to degrees
50  PRINT "RAD.=";RAD                     :'Display
60  PRINT "DEG.=";DEG
70  END

RUN
RAD.=1.04719
DEG.=59.9993
OK
```

REMARK

See COS, SIN, TAN.

- Automatically displays line numbers at the beginning of lines.

AUTOmatic

SYNTAX	**AUTOΔ<line number>, <increment>**

	line number	specifies the starting line number.
	increment	Specifies the increment between line numbers.
EXAMPLE	**AUTO**	Starting with line number 10, generate line numbers in increments of 10. (line number 10, 20, 30, ...)
	AUTO 100	Starting with line number 100, generate line numbers in incrementsof 10. (line number 100, 110, 120, ...)
	AUTO 100, 20	Starting with line number 100, generate line numbers in increments of 20. (line number 100, 120, 140, ...)
	AUTO , 20	Starting with line number 0, generate line numbers in increments of 20. (line number 0, 20, 40, ...)
	AUTO 100	Starting with line number 100, generate line numbers with the same increment as the previously executed AUTO command.

DESCRIPTION

- The **AUTO** command displays line numbers automatically according to the user's specifications.
- The value indicated by **<line number>** becomes the first line number. After this, each successive line number is incremented by the amount specified in **<increment>**.
- If both **<line number>** and **<increment>** are omitted, the values are assumed to be 10.
- If **<line number>** only is omitted, the value is assumed to be 0.
- If the **<line number>** is specified, but after the comma is omitted, the increment specified in the previously executed **AUTO** command is assumed.
- If the **AUTO** command displays a line number that has already been used, an asterisk is displayed following the line number. If no characters are entered following the asterisk, simply pressing the **[Enter]** key will preserve the original contents of the line.
- To end execution of the **AUTO** command, press **[Ctrl]**+**[C]** or **[Break]**.
- While the **AUTO** command is executing, it is not possible to edit a previously entered line by moving the cursor with the cursor keys.

REMARK

See section 3.1.

- Produces a beep sound from the console's internal speaker.

BEEP

SYNTAX	**BEEP**
EXAMPLE	BEEP

DESCRIPTION

- The **BEEP** command works the same as when **PRINT CHR$**(7) is executed; a tone isproduced for a predefined length of time.
- The **BEEP** command only affects the port that is specified for the console by the AD51H system dip switch. The **ZODV** command cannot be used to change the output destination.

PROGRAMMING EXAMPLE

```
10 'Sound the beep 10 times.
20 FOR I=1 TO 10                    :'Loop 10 times
30  BEEP                            :'Sound the beep
40  PRINT I;"Times"                 :'Print loop number
50   FOR J=0 TO 200                 :'Use an empty loop to kill time
60   NEXT J
70 NEXT I
80 END
```

- Returns the binary representation of an integer as a character string.

BINary $

SYNTAX	**BIN$ (<numeric expression>)**

	numeric expression	Numbers in the range -32768 to 65535 can be specified. If the number specified is in the range 32768 to 65535, the results are the same as for numbers in the range -32768 to -1 (8000_H to $FFFF_H$).
EXAMPLE	**A$ = BIN$(10)**	The character string representing the integer 10 in binary is stored to A$.

Decimal → Binary
10 → 1010

DESCRIPTION

- The **BIN$** function returns the binary representation of an integer as a character string.
- If the value specified for **<numeric expression>** contains figures following the decimal point, the number is converted to an integer by truncating all numbers following the decimal point.

POINTS

(1) Integers that can be handled with 16 bits are in the range -32768 to 32767. The hexadecimal representation of these numbers is shown below.

(2) If a base 10 number in the range 32768 to 65535 is specified, it is considered to be the same as if the corresponding number in the range -32768 to -1 had been specified.

Decimal	Hexadecimal	Binary				
65535 to 32768	$FFFF_H$ to 8000_H	1111 1000	1111 0000	to	1111 0000	1111 0000
32767 to 0	$7FFF_H$ to 0000_H	0111 0000	1111 0000	to	1111 0000	1111 0000
-1 to -32768	$FFFF_H$ to 8000_H	1111 1000	1111 0000	to	1111 0000	1111 0000

Handled identically

PROGRAMMING EXAMPLE

```
10 'Convert a base 10 number to a binary (base 2) number.
20 A=1234                                    :'Define the numeric value
30 A$=BIN$(A)                                :'Convert to binary
40 PRINT "Decimal (base 10)=";A
50 PRINT "Binary (base 2)=";A$
60 END

RUN
Decimal (base 10)= 1234
Binary (base 2)=10011010010
OK
```

REMARKS

See VAL, HEX$ and OCT$.

- Interchanges the byte position of two values.

Byte SWAP

SYNTAX	**BSWAPΔ<variable 1>,<variable 1 interchange start address>, <variable 2>,<variable 2 interchange start address>, <number of bytes>**

variable 1, variable 2 — Specifies the variables on which to perform the interchange operation.

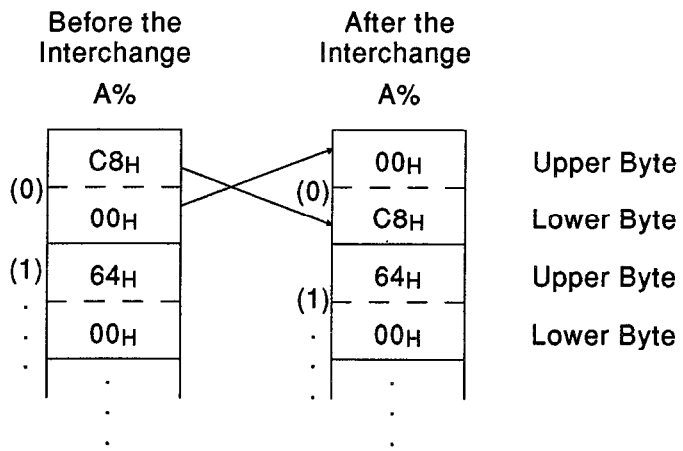
variable 1, variable 2 interchange start address — Specifies the address to begin the interchange operation interchange start address for each variable.

number of bytes — Specifies how many bytes to interchange

EXAMPLE

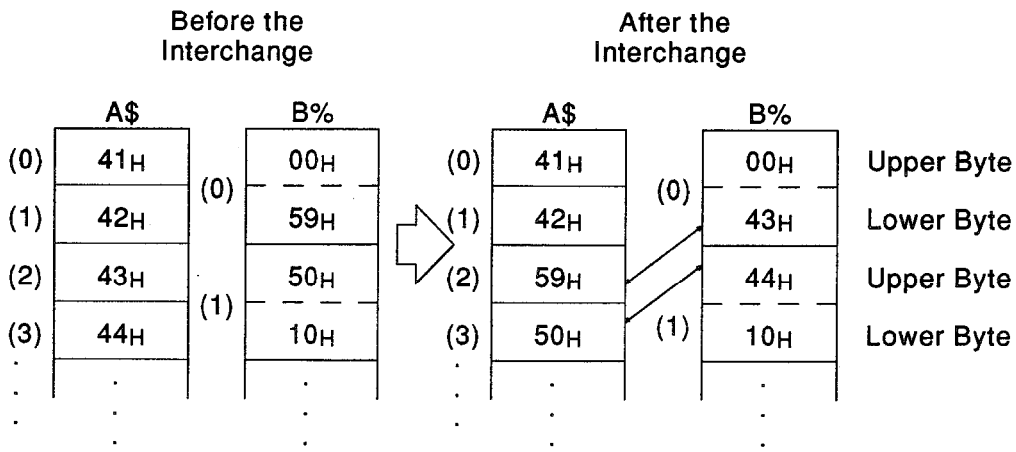
BSWAP A%( ), 0, A%( ), 1, 1

The upper byte of A%(0) is interchanged with the lower byte.



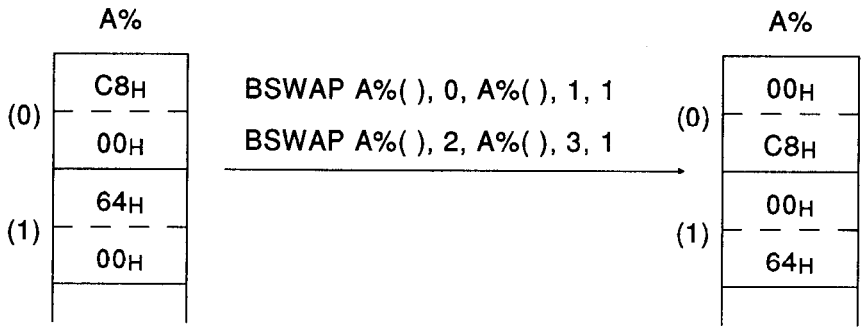
BSWAP A$, 3, B%( ), 1, 2

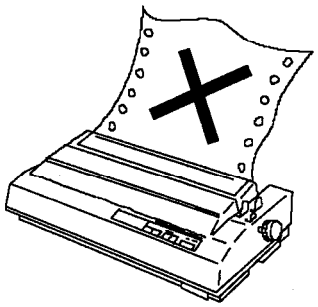
The two bytes starting with character array A$(2) are interchanged with the two bytes starting with the lower byte of integer array B%(0).

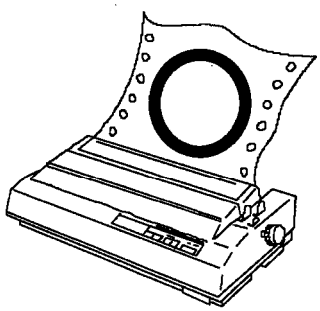


DESCRIPTION

- The **BSWAP** command exchanges the values of two variables beginning at a specified address within the variables.
  This command is used especially when communicating with an external device to convert data to the proper format for the device being used.







- If the starting address specified for an array variable is bigger than the size of the variable as specified in the **DIM** statement, an error occurs.
- The same variable can be specified as both **<variable 1>** and **<variable 2>**.
- **<Number of bytes>** specifies how many bytes to interchange.

PROGRAMMING EXAMPLE

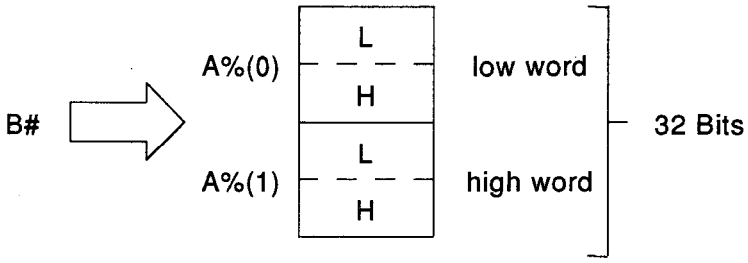
```
10 'Interchange the contents of array elements A%(0) and A%(1).
20 DIM A%(1)                                  :'Define the array
30 A%(0)=&H924                                :'Define the array's values
40 A%(1)=&H1159
50 PRINT "Before Interchange"                 :'Display before interchange
60 PRINT "A%(0)=&H";HEX$(A%(0))
70 PRINT "A%(1)=&H";HEX$(A%(1))
80 BSWAP A%( ), 0, A%( ), 2, 2                :'Interchange
90 PRINT "After Interchange"                  :'Display after interchange
100 PRINT "A%(0)=&H";HEX$(A%(0))
110 PRINT "A%(1)=&H";HEX$(A%(1))
120 END

RUN
Before Interchange
 A%(0)=&H924
 A%(1)=&H1159
After Interchange
 A%(0)=&H1159
 A%(1)=&H924
OK
```

- Converts a double precision real number to a 2 word (32 bit) integer for use by the PC CPU.

Convert DouBle to Integer

SYNTAX	**CDBI△<precision real number variable>,<array variable>**

	double precision real number variable	Specifies the variable that contains the data to be converted.
	array variable	Specifies the one dimensional integer array to contain the converted data.
EXAMPLE	**CDBI B#, A% (0)** ⟶	The value of B# is converted to an integer and stored to A%(0) and A%(1).

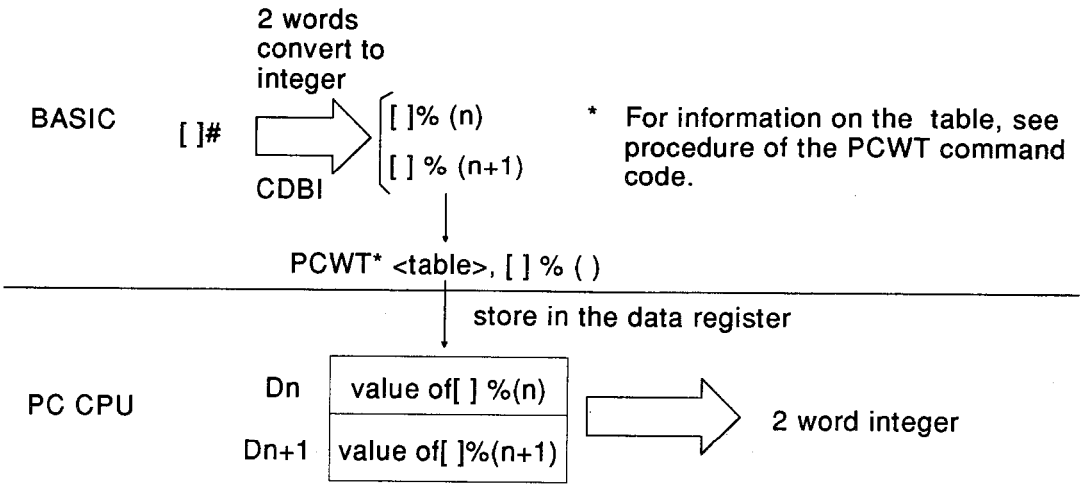


DESCRIPTION

- The **CDBI** command converts a double precision real number to a 2 word (32 bit) integer for use by the PC CPU.
- Double precision real numbers that can be handled in BASIC are in the range -2147483648 to 2147483647.
  If a value outside this range is specified, an **"Overflow"** error occurs.
  If the double precision value includes figures following the decimal point, it will be truncated at the decimal point before it is converted.

- The following illustrates how the PC CPU handles BASIC double precision real numbers.

Example



- A one dimensional integer array must be used for **<array variable>**. If any type of variable other than a one dimensional integer array is used, an **"Illegal function call"** error occurs.

POINT

The variable used as array variable must be defined with the DIM command.
If the variable is not defined with the DIM command, an "Illegal function call" error occurs.

PROGRAMMING EXAMPLE

```
10 'Convert a double precision real number to a two word integer and write it to D0,D1
20 DIM B%(1), TBL%(5)                    :'Define the array
30 A#=1234567890#                        :'Define the double precision real value
40 CDBI A#, B%(0)                        :'Convert to a 32 bit integer
50 TBL%(0)=255                           :'Set the communication station to the host
60 TBL%(1)=1                             :'Specify writing to device memory
70 TBL%(2)=2                             :'Specify word units
80 TBL%(3)=18                            :'Specify data range
90 TBL%(4)=0                             :'Specify device number
100 TBL%(5)=2                            :'Specify operation point count
110 PCRD TBL%( ), B%( )                  :'Execute write operation
120 END
```

REMARK

See the CDBI, CIDB, CISN, PCRD and PCWT commands.
Also see chapter 4.

CDBL function

- Converts an integer or a single precision real number to double precision real number.

Convert to DouBLe

SYNTAX	**CDBL (<numeric expression>)**	
	numeric expression	Specifies the integer or single precision real number which is to be converted to a double precision real number.
EXAMPLE	**A#=CDBL (B !)**	The value of single precision real variable B! is converted to a double precision real number and stored to A#.
	C#=CDBL (D%)	The value of integer variable D% is converted to a double precision real number and stored in C#.

DESCRIPTION

- The **CDBL** function converts the value of **<numeric expression>** to a double precision real number.
- The value's format is changed, but the significant digits are not changed.
- The precision of the converted value is the same as before the conversion (for integers, only the integer part is precise, for single precision real numbers,only the first six significant figures).
- When a single precision real number is converted with the **CDBL** function,sometimes the original single precision value and the converted value are not exactly the same. This is because some values cannot be correctly represented internally.

Example

```
PRINT CDBL(0.1!)
.1 0 0 0 0 0 0 0 1 4 9 0 1 1 6 1
OK
■
```

PROGRAMMING EXAMPLE

```
10  'Convert an integer and a single precision real number to a double precision real
    number.
20  A%=256                                  :'Define an integer
30  B!=5.78                                 :'Define a single precision real number
40  A#=CDBL(A%)                             :'Convert the integer to a double precision
                                             real number
50  B#=CDBL(B!)                             :'Convert the single precision real number
                                             to a  double precision real number
60  PRINT "A%=";A%,"B!=";B!                 :'Pre-conversion values
70  PRINT "A#=";A#,"B#=";B#                 :'Post-conversion values
80 END

RUN
A%= 256              B!= 5.78
A#= 256              B#= 5.779999732971191
OK
```

REMARK

When a value is assigned to a double precision variable, or if a value is used as a double precision value in part of an operation, the value is automatically converted to double precision format. Therefore, the statement A# = CDBL (3042.12!) produces the same result as A# = 3042.12!.

- Delete all or part of the currently executing program, and read in and execute the specified program.

CHAIN

SYNTAX	**CHAINΔ"[<drive number>:][<system name>\] <file name>" [, <number 1>][, ALL]** **CHAINΔMERGEΔ"[<drive number>:][<system name>\] <file name>" [, <line number 1][, ALL][, DELETEΔ<line number 2> - <line number 3>]**

drive number	Specifies the disk or memory card where the program to be read in is located.
system name	Specifies the system name where the program to be read in is located.
file name	Specifies the file name of the program to be read in.
line number 1	Specifies the line number at which to begin program execution.
line number 2	Specifies the range of lines to delete with the DELETE option.
line number 3	Specifies the range of lines to delete with the DELETE option.

EXAMPLE

CHAIN "0:PRGM-A"

The currently executing program is completely cleared, the program stored in file "PRGM-A.BAS" on the memory card inserted in M-CARD 0 is read in, the variables specified with the COMMON command are passed, and execution is started.

CHAIN "1:PRGM-A",200,ALL

The currently executing program is completely cleared, the program stored in file "PRGM-A.BAS" on drive 1 is read in, all variables, arrays, etc. are passed, and execution is started beginning at line 200 of the program that was read in.

CHAIN MERGE "A:SUB-A" DELETE 1000-2000

Lines 1000 2000 of the currently executing program are deleted, the program stored in file "SUB-A.BAS" on drive 0 is read in and combined with the existing program, and execution is started beginning at line 1000.

CHAIN MERGE "0:TEST",1000,DELETE 1000-1500

Lines 1000 1500 of the currently executing program are deleted, the program stored in file "TEST.BAS" on drive 0 is read in and combined with the existing program, and execution is started beginning at line 1000.

DESCRIPTION

- The **CHAIN** command reads a program (new program) into the same BASIC number area that the program currently in memory (old program) is located. Then the contents of the variables in the currently executing program (old program) are passed to the new program, and the new program is executed.
- **<Drive number>** : **<system name>** **<file name>** are used to specify the new program to be read in. For details, see section 3.3.1.
- If **<line number 1>** is specified, execution of the newly read-in program is begun starting from the specified line number. If this is omitted, execution starts from the first line of the program.
- The following two methods are used to pass variables, arrays, etc. from the current program (old program) to the newly read-in program (new program).

  (1) If the **ALL** option of the **CHAIN** command is specified, all variables, arrays,etc. are passed from the old program to the new program.

  (2) To pass only some variables, arrays, etc. to the new program, do not specify the **ALL** option in the **CHAIN** command. Instead, omit the **ALL** option, and use the **COMMON** command to specify which variables, arrays, etc. to pass.
- The following points should be noted when passing variables.

  (1) User defined functions defined with the **DEF FN** command cannot be passed.

  (2) The format of constants defined with the **DEFDBL, DEFSNG, DEFINT** and **DEFSTR** commands can only be passed when the **MERGE** option is specified.

WHEN THE MERGE OPTION IS SPECIFIED:

- When the merge option is specified, the new program is combined with the old program and executed. This command makes it possible for internal subroutines to be moved around. To change the program's contents, it is necessary to specify which line numbers of the old program will be deleted before the new program is inserted. This is done using the **DELETE** option. Specifically, the lines in the range **<line number 2> - <line number 3>** are deleted from the old program, and the new program is inserted in the deleted space.
  When **MERGE** and **DELETE** are specified, depending on the line numbers of the old program and the new program, the lines may be re-arranged.
- If the **DELETE** option is specified, the **MERGE** option must also be specified.

REMARK

See section 3.15.

- Returns the character corresponding to the character code specified as a numeric expression.

CHaRacter $

SYNTAX	**CHR$ (<numeric expression>)**	
	numeric expression	Any number in the range 0 to 255 (hexadecimal: &H00 to &HFF) can be specified.
EXAMPLE	**PRINT CHR $ (&H41)**	Displays the character "A" which corresponds to character code 41H.

DESCRIPTION

- The **CHR$** function returns the character corresponding to the character code specified as **<numeric expression>**. expression must be in the range 0 to 255.
- The **CHR$** function is used to display both normal and special characters.
- For information on converting a character to a numeric value, see the **ASC** function.

PROGRAMMING EXAMPLE

```
10 'Display the characters corresponding to character codes &H41 to &H5A
20 FOR I=&H41 TO &HSA                          :'Loop from &H41 to &H5A
30  PRINT CHR$<I>;" ";                         :'Display the character
40 NEXT I
50 END


RUN
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
OK
```

REMARK

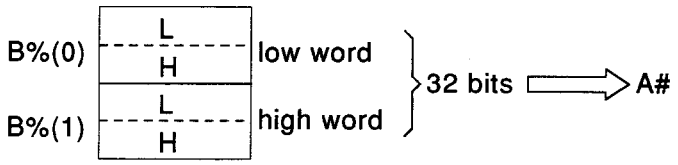
See ASC.

- Converts a two word integer used by the PC CPU into a double precision real number.

Convert Integer to DouBle

SYNTAX	**CIDB (array variable)**

	array variable	Specifies the one dimensional integer array which contains the data to convert.
EXAMPLE	**A#=CIDB(B%(0))**	The integer value stored in B%(0) and B%(1) is converted into a double precision real number and assigned to A#.



DESCRIPTION

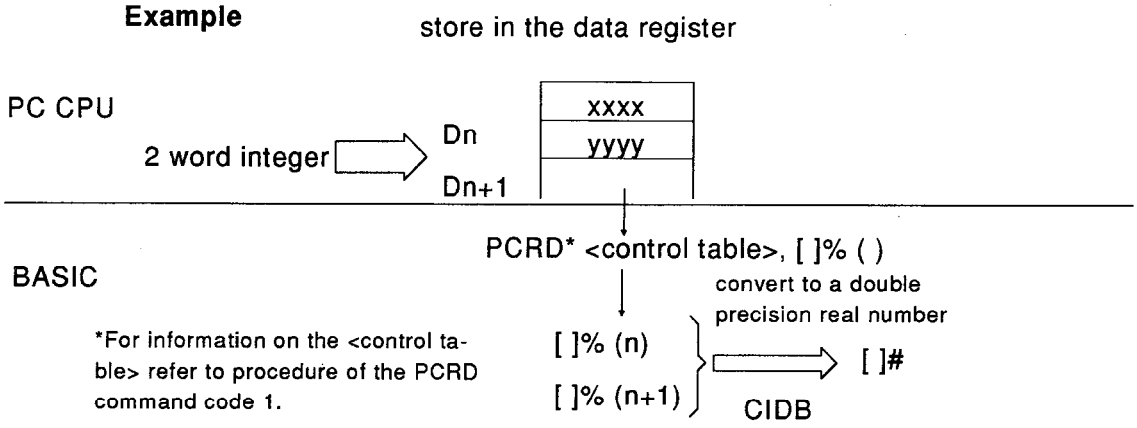
- The **CIDB** function converts a two word integer used by the PC CPU into a double precision real number.
- A one dimensional integer array must be specified for **<array variable>**. If any type of variable other than a one dimensional integer array is specified, an **"Illegal function call"** error occurs.

POINT

The variable specified for array variable must be defined with the DIM command. If it is not defined with the DIM command, an "Illegal function call" error occurs.

- The precision of a double precision real number that is converted from a two word (32 bit) integer is the same as before the **CIDB** function is executed.
- The two word (32 bit) integers used by the PC CPU are handled by BASIC as illustrated below.

Example



PROGRAMMING EXAMPLE

```
10 'Convert the two word integer stored in D0, D1 to a double precision real number
20 DIM TBL%(5), A%(1)                          :'Define the array
30 TBL%(0)=255                                 :'Set the communication station number to the host
40 TBL%(1)=1                                   :'Specify reading from device memory
50 TBL%(2)=2                                   :'Specify word units
60 TBL%(3)=18                                  :'Specify the data register
70 TBL%(4)=0                                   :'Specify the read device head number
80 TBL%(5)=2                                   :'Specify the number of points to read
90 PCRD TBL%( ), A%( )                         :'Execute read
100 A#=CIDB(A%(0))                             :'Convert to a double precision real number
110 PRINT "The value of D0, D1 is -->";A#
120 END
```

REMARK

See CSNI, CIDB, CISN, PCRD and PCWT. Also see chapter 4.

- Converts a single or double precision real number value to an integer value.

Convert INTeger

SYNTAX	**CINT (<numeric expression>)**	
	numeric expression	Specifies the real number to be converted to an integer.
EXAMPLE	**A%=CINT (B !)**	The value of single precision real variable B! is converted to an integer value and assigned to A%.
	C%=CINT (D#)	The value of double precision real variable D# is converted to an integer value and assigned to C%.

DESCRIPTION

- The **CINT** function converts the value of **<numeric expression>** to an integer and returns as its value the largest integer that is smaller than the value of expression.
- If a positive number is specified, the number is truncated at the decimal point. If a negative number is specified, it is rounded up to the next whole number.

Example

5.689	⟶ 5	-3.84	⟶ -4
1.031	⟶ 1	-1.2639	⟶ -2
2.999	⟶ 2	-1.9999	⟶ -2

- If the result of the conversion exceeds the range -32768 to 32767, an **"Overflow"** error occurs. To convert values greater than 32767, use the **INT** function.

PROGRAMMING EXAMPLE

```
10 'Convert a single and a double precision real number to integers
20 A!=7.15                                  :'Define the single precision real number
30 B#=1.598421#                             :'Define the double precision real number
40 A%=CINT(A!)                              :'Convert the single precision real number
                                             to an integer
50 B%=CINT(B#)                              :'Convert the double precision real number
                                             to an integer
60 PRINT "A!=";A!,"B#=";B#                  :'Pre-conversion values
70 PRINT "A%=";A%,"B%=";B%                  :'Post-conversion values
80 END

RUN
A!= 7.15        B#= 1.598421
A%= 7           B%= 1
OK
```

REMARK

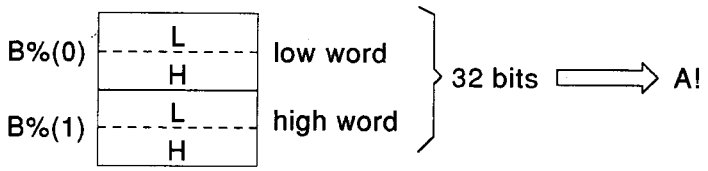
When a value is assigned to an integer type variable or is used as a parameter (argument) for a command or function, it is automatically converted to an integer value. The statements **A% = 3042.12!** and **A% = CINT(3042.12!)** produce the same results.

- Converts a two word integer used by the PC CPU to a single precision real number.

Convert Integer to SiNgle

SYNTAX	**CISN (<array variable>)**

	array variable	Specifies a one dimensional integer array that contains the data which is to be converted.
EXAMPLE	**A !=CISN (B%(0))**	Convert the integer value stored in B%(0) and B%(1) to a single precision real value and store it in A!.



DESCRIPTION

- The **CISN** function converts a two word integer used by the PC CPU to a single precision real number.
- A one dimensional integer array must be specified for **<array variable>**. If any type of variable other than a one dimensional integer array is specified, an **"Illegal function call"** error occurs.

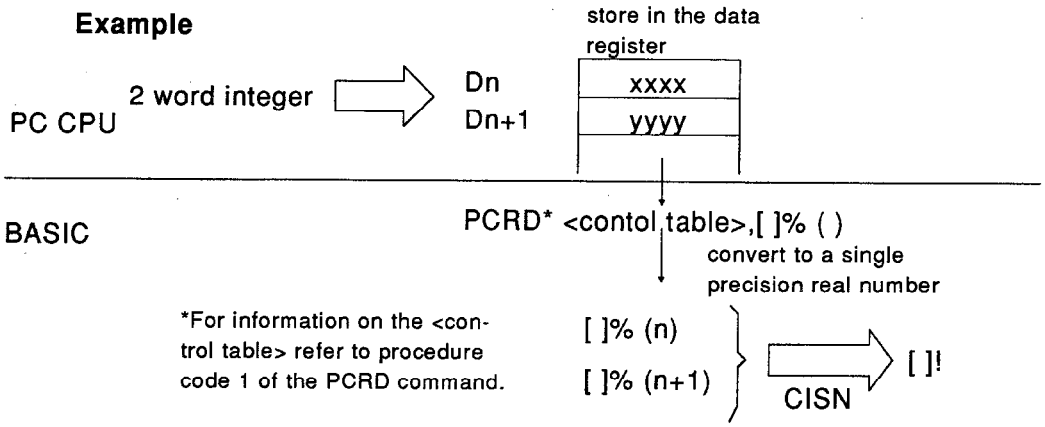
POINT

The variable specified for array variable must be defined with the DIM command. If it is not defined with the DIM command, an "Illegal function call" error occurs.

- The precision of a single precision real number that is converted from a two word (32 bit) integer is the same as before the **CISN** function is executed.

The two word (32 bit) integers used by the PC CPU are handled by BASIC as illustrated below.

Example



PROGRAMMING EXAMPLE

```
10 'Convert the two word integer stored in D0, D1 to a single precision real number
20 DIM TBL%(5),A%(1)                          :'Define the array
30 TBL%(0)=255                                :'Set the communication station number to
                                               the host
40 TBL%(1)=1                                  :'Specify reading from device memory
50 TBL%(2)=2                                  :'Specify word units
60 TBL%(3)=18                                 :'Specify the data register
70 TBL%(4)=0                                  :'Specify the read device head number
80 TBL%(5)=2                                  :'Specify the number of points for the
                                               operation
90 PCRD TBL%( ),A%( )                         :'Execute read
120 A!=CISN(A%(0))                            :'Convert to a single precision real number
140 PRINT "The value of D0, D1 is -->";A!
150 END
```

REMARK

See CIDB, CDBI, CSNI, PCRD and PCWT. Also see chapter 4.

- Initializes all variables and sets up a memory range.

CLEAR

SYNTAX	**CLEARΔ[<numeric expression>]**	
	numeric expression	Specifies the size of the memory range in byte units to be used by BASIC to store character strings. If this is omitted, the default value 300 is assumed.
EXAMPLE	**CLEAR 1000**	Initialize all variables and sets up a 1000 byte range of memory.

DESCRIPTION

- The **CLEAR** command initializes all variables and sets up a memory range.
- When the **CLEAR** command is executed, the contents of variables, declarations and definitions are set to the following values:

  Integer variables 0

  Character variables ..... empty string

  Array definitions (DIM statement) ..... invalid

  Variable type declarations .....invalid

  User defined functions ..... invalid

- If many character strings are to be managed, a large memory range is necessary. If the memory range is not sufficient, an **"Out of string space"** error occurs. If this occurs, the argument of the **CLEAR** command can be set to increase the size of the memory range.

REMARK

The ERASE command can be used to initialize only array variables.

See DIM, FRE.

- Completes a file input/output operation.

CLOSE

SYNTAX	**CLOSEΔ<<#>file number>,<<#>file number>...>**

	file number	Must be in the range 1 to 8. Indicates a file number that was opened with the OPEN command.
EXAMPLE	**CLOSE**	All files are closed and all file input/output operations are completed.
	CLOSE #1	File #1 is closed and input/output operations on the file corresponding to file #1 are completed.

DESCRIPTION

- The **CLOSE** command completes data input/output operations and closes (end procedure) a file.
- **<File number>** indicates the number of a file that was opened with the **OPEN** command.
- If **<file number>** is omitted, all currently open files are closed.
- If the **RUN, CLEAR, END, NEW, LOAD, CLOSE** or **SYSTEM** commands are executed, all files are automatically closed.
  The **STOP** command, however, does not cause files to be closed.

REMARK

See OPEN, END, STOP, NEW, LOAD, and CHAIN. Also see chapter 6.

- Clears the screen.

CLear Screen

SYNTAX	**CLS[<function>]**

	function	This should be set to "1".
EXAMPLE	**CLS** **CLS 1**	The text screen is cleared.

DESCRIPTION

- The **CLS** command clears the console screen specified with the **ZODV** command and moves the cursor to the top, left of the screen.
- **<Function>**should be set to 1. If it is omitted, the value 1 is assumed.

REMARK

See section 3.10.2.

- Enables/disables or stops input from the communication line.

COMmunication ON/OFF/STOP

SYNTAX	**COM [(channel number)] ON** **COM [(channel number)] OF** **COM [(channel number)] STOP**	
	channel number	Specifies the communication port where communication line input is to be controlled.
EXAMPLE	**COM (1) ON**	Enables input via CH1 (RS-232C).

DESCRIPTION

- The **COM ON/OFF/STOP** command enables (**ON**), disables (**OFF**), or stops (**STOP**) input from an external source through the communication ports (RS-232C/RS-422).
- **<Channel number>**specifies the communication port through which communication line input is performed.
  The following shows the corresponding communication port for each **<channel number>.**

  1 ..... CH1 (RS-232C)

  2 ..... CH2 (RS-232C)

  3 ..... CH3 (RS-422)
- **<Channel number>** can be omitted. If it is omitted, both the RS-232 ports and the RS422 port are specified.
- Immediately after the **COM ON** command is executed, communication line input is enabled. After this command has been executed, when input is received through the communication port specified by **<channel number>**, execution is diverted to the communication routine defined with the **ON COM GOSUB** command.
- If the **COM OFF** command is executed, communication line input is disabled. After this command is executed, even if input is received at the communication porexecution is not diverted to the communication routine.
- When the **COM STOP** command is executed, communication line input is stopped. After this command is executed, input received via the communication port is stored in memory only. Execution is not diverted to the communication routine. If the **COM ON** command is later executed, input is enabled, and execution is diverted to the communication routine.
- Before ending program execution, the communication port should be in **COM OFF** condition.
- Under multitasking, each port can be accessed by only one program.

REMARK

See ON COM GOSUB and ZOPEN. Also see section 7.4 and appendix 2.

- Defines the variables that will be passed when the CHAIN command is used to execute a new program.

COMMON

SYNTAX	**COMMONΔ<data to pass> [,<data to pass>, ···]**	
	data to pass	Specifies variables or arrays to be passed when the CHAIN command is used to execute a new program.
EXAMPLE	**COMMON A$,B%( )**	Passes the variable A$, and all elements of the array B%( ) to the program executed with the CHAIN command.

DESCRIPTION

- The **COMMON** command selects the variables and arrays to be passed to the program executed by the **CHAIN** command.
- <Data to pass> specifies the variables or arrays to pass to the program that is read in and executed with the **CHAIN** command.
- When an array is specified, only empty parentheses should be written after the array name.
  A% ( ), B# ( ), C$ ( )
- If there is a mistake in the **COMMON** command, when the **CHAIN** command is executed,an error occurs.
- To pass all variables and arrays to the new program, the **ALL** option of the **CHAIN** command is used.
- If <data to pass> is set with the **COMMON** command, but then the **ALL** option is specified in the **CHAIN** command, all variables and arrays will be passed to the new program regardless of the settings made with the **COMMON** command.

REMARK

The COMMON command may be included in a part of a program.

The COMMON command should be included in the program that executes the CHAIN command.

See CHAIN.

- Sets the number of lines that can be displayed on the console screen.

CONSOLE

SYNTAX	**CONSOLE <number of lines >**	
	number of lines	Specifies the number of lines (in the range 20 to 32) that can be displayed by the console in use.
EXAMPLE	**CONSOLE 25**	Specifies a 25 line display.

DESCRIPTION

- The CONSOLE command sets the number of display lines on the console in order to be compatible with many different kinds of console screens.
- The default value of the **CONSOLE** is set in accordance with the console set in MODE SW2.
- The following indicates the number of lines that should be set for various consoles.

  VG-620/VT-386/VT-220 ..... 24 lines

  IBM PC/AT ..... 25 lines

- If the number of lines specified is different from the actual number of lines the console screen can display, scrolling cannot function properly.

- In edit mode, restarts execution of a program that has been stopped with the [Ctrl]+[C] [Break] keys, or with the STOP or END commands.

CONTinue

SYNTAX	**CONT**

EXAMPLE	**CONT**	Restarts execution of a stopped program from the point where it was stopped.

DESCRIPTION

- The **CONT** command restarts execution of a stopped program.
- Execution is started from the line following the line where execution was stopped.
- If the program was stopped while waiting for input for the **INPUT** command, the question mark is re-displayed when execution is restarted.
- The **CONT** command is normally used with the **STOP** command to aid debugging. When program execution is stopped, intermediate results can be checked, and values changed via direct mode. Then the **CONT** or **GOTO** commands can be used to restart execution.
- If the **CONT** command is executed after the program has been changed, a **"Can't continue"** error occurs. For more information on debugging methods, see chapter 10.

POINTS

(1) The CONT and STOP commands are only valid in edit mode.

(2) When program execution is started with the RUN command, the contents of all variables and definitions are cleared.

If program execution is completed by an END command which appears in the middle of the program, the CONT command can be used to restart execution from the line following the END statement.

REMARK

See END, RUN, and STOP. Also see chapter 10.

- Calculates the value of the trigonometric function cosine (COS).

COS

SYNTAX	**COS (<numeric expression>)**	
	numeric expression	Specifies a numeric value or numeric value variable in radian units.
EXAMPLE	**A=COS(3.14159/180*60)**	The angle measure 60 degrees is specified in radians,the cosine is calculated, and the result is stored in A.

DESCRIPTION

- The **COS** function returns the cosine of the **<numeric expression>** specified inradian units ((π /180) × dgree measure).
- The **<number expression>** can be specified in any desired precision format, but the **COS** function performs its calculations in single precision.

PROGRAMMING EXAMPLE

```
10  'Calcluate cos 60°
20  A=(3.141592653#*60)/180          :'Convert 60° to radians
30  B=COS(A)                         :'Calculate the cosine
40  PRINT "60°=";A;"radians"
50  PRINT "cos60°=";B

RUN
60°=1.0472 radians
cos°60= .5
OK
```

REMARK

See ATN, SIN and TAN.

- Converts an integer or double precision real number to a single precision real number.

Convert SiNGle

SYNTAX	**CSNG (<numeric expression>)**	
	numeric expression	Specifies the integer or double precision real number to be converted to a single precision real number.
EXAMPLE	**A!=CSNG(B%)**	The value of integer variable B% is converted to a single precision number and stored in A.
	C!=CSNG(D#)	The value of double precision real variable D# is converted to a single precision number and stored in C.

DESCRIPTION

- The **CSNG** function converts the value of **<numeric expression>** to a single precision real number with six_significant figures.
- If the converted value is outside of the range -1.70141E+38 to 1.7014E+38, an **"Overflow"** error occurs.

PROGRAMMING EXAMPLE

```
10  'Convert an integer and a double precision real number to a single precision real
    number.
20  A%=2                                    :'Define an integer
30  B#=1.37825432#                          :'Define a double precision real number
40  A!=CSNG(A%)
    :'Convert the integer to a single precision real number
50  B!=CSNG(B#)
    :'Convert the double precision real number to a siprecision real number
60  PRINT "A%=";A%,"B#=                     ";B#:'Pre-conversion values
70  PRINT "A!=";A!,"B!=                     ";B!:'Post-conversion values
80  END

RUN
A%= 2          B#= 1.37825432
A!= 2          B!= 1.37825
OK
```

REMARK

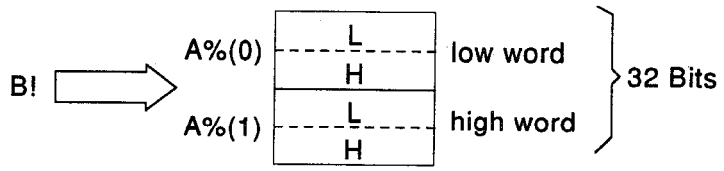
When a value is assigned to a single precision variable, or if a value is used as a single precision real value parameter (argument) for a command or function, the value is automatically converted to single precision format. Therefore, the statement A! = CSNG (3042.1545452#) produces the same result as A! = 3042.1545452#.

- Converts a single precision real number to a 2 word (32 bit) integer for use by the PC CPU.

Convert SiNgle to Integer

SYNTAX	**CSNIΔ<single precision real number variable>, <array variable>**

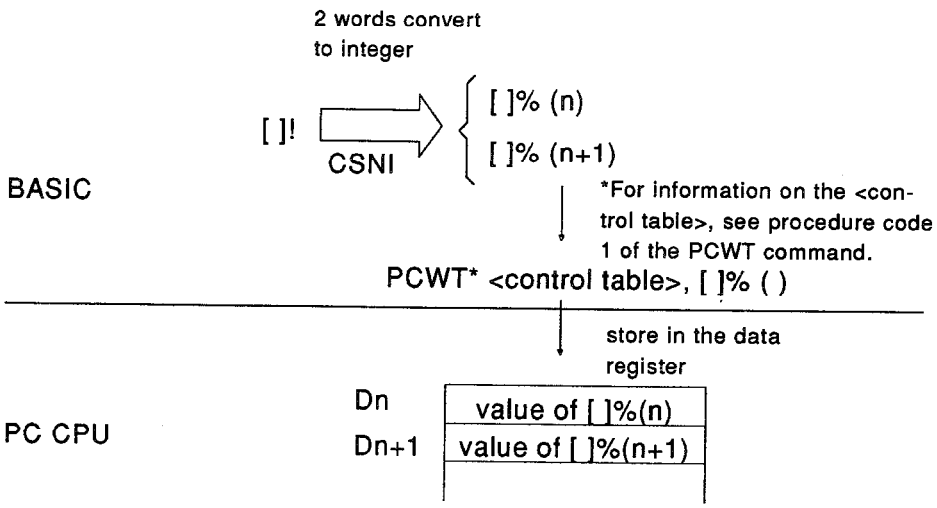
	single precision real number variable	Specifies the single precision real variable that contains the data to be converted.
	Array variable	Specifies the one dimensional integer array to contain the converted data.
EXAMPLE	**CSNI B!, A% (0)** →	The value of B! is converted to an integer and stored to A%(0) and A%(1).



DESCRIPTION

- The **CSNI** command converts a single precision real number to a 2 word (32-bit) integer for use by the PC CPU.
- Single precision real numbers that can be handled in BASIC are in the range -2.14748E+09 to 2.14748E+09. If a value outside this range is specified, an **"Overflow"** error occurs. If the single precision value includes figures following the decimal point, it will be truncated at the decimal point before it is converted.
- The following illustrates how the PC CPU handles BASIC single precision real numbers.

Example



- A one dimensional integer array must be used for <array variable>. If any type of variable other than a one dimensional integer array is used, an "Illegal function call" error occurs.

POINT

The variable used as array variable must be defined with the DIM command. If the variable is not defined with the DIM command, an "Illegal function call" error occurs.

PROGRAMMING EXAMPLE

```
10 'Convert a single precision real number to a two word integer and write it to D0,D1
20 DIM B%(1)                                  :'Define the array
30 A!=999999!                                 :'Define the single precision real value
40 CSNI A!, B%(0)                             :'Convert to a 32 bit integer
70 TBL%(0)=255                                :'Set the communication station number to
                                               the host
80 TBL%(1)=1                                  :'Specify writing to device memory
90 TBL%(2)=2                                  :'Specify word units
100 TBL%(3)=18                                :'Specify data register D
110 TBL%(4)=0                                 :'Specify device number (0)
120 TBL%(5)=2                                 :'Specify operation point count
130 PCRD TBL%( ), B%( )                       :'Execute write operation
140 END
```

REMARK

See the CISN, CDBI, PCRD and PCWT commands. Also see chapter 4.

- Convert a character string created with the MKD$ function back to a double precision real number.

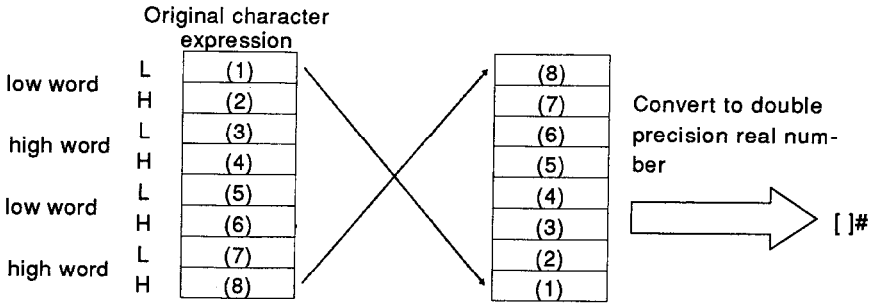
ConVert to Double

SYNTAX	**CVD (<character string expression> [,s])**

	character string expression	Specifies the string that was created with the MKD$ function.
EXAMPLE	**A#=CVD (MD$)**	A string (MD$) created with the MKD$ function is converted back to the original double precision real number value and assigned to A#.
	B#=CVD (MD$,S)	The contents of character string MD$ are re-arranged, and then the string is converted back to the original double precision real number value. The value is assigned to B#.

DESCRIPTION

- The **CVD** function converts a character string created with the **MKD$** function back to the original double precision real number.
- Since all data written to random files must be character strings, the **MKD$** function is used to convert double precision real number values to 8 byte character strings so they can be written to the file. When this data is read back from the file, it is necessary to convert it back to the original double precision real number value. The **CVD** function is used for this purpose.
- The **CVD** function can convert only character strings that were converted from a double precision real number value using the **MKD$** function. If any other character string is specified for **<character string expression>**, meaningful results will not be produced, or an **"Illegal function call"** error may occur.
- Data converted to a character string with the **MKD$** function is not only used for writing to a random file, it can also be used for data communication.
- If the **[,s]** option is specified, the character string to be converted is first re-arranged as shown below. Following the re-arrangement, it is converted into a double precision real number value.



REMARK

See MKD$.

- Converts a string that was converted from an AD51H-BASIC double precision real number with the MKD$ function into IEEE format double precision internal representation.

Convert Double Microsoft Binary Format

SYNTAX	**CVDMBF (<character string expression>)**	
	character string expression	Specifies a string that was converted from a double precision real number with the MKD$ function.
EXAMPLE	**A#=CVDMBF (MKD$(B#))**	Convert the double precision real number value stored in B# to IEEE format internal representation and assign it to A#.

DESCRIPTION

- The **CVDMBF** function converts a string that was converted from an AD51H-BASIC double precision real number with the **MKD$** function into IEEE format double precision internal representation.
- The AD51H-BASIC internal representation of real numbers, and the IEEE internal representation of floating point real numbers are different. The **CVDMBF** function is used to convert between the two representations.
- **<Character string>** argument should always be a string that was converted from a double precision real number with the MKD$ function. Strings created with the **MKS$** or **MKI$** functions cannot be converted correctly.
- The value returned by **CVDMBF** is a double precision real number. Therefore, this value should be assigned only to double precision real type variables ([ ]#).
- Values converted with the **CVDMBF** function can be used when transferring data to a system which uses the IEEE format for internally representing double precision real numbers.

REMARKS

(1) The value returned by the CVDMBF function has no meaning within AD51H-BASIC.

(2) See CVSMBF, MKDMBF$, and MKSMBF$.

- Convert a character string created with the MKI$ function back to an integer.

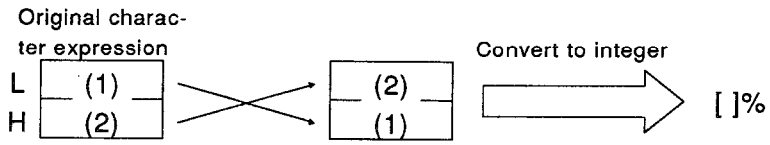
ConVert to Integer

SYNTAX	**CVI (<character string expression> [,s])**

	character string expression	Specifies the string that was created with the MKI$ function.
EXAMPLE	**A%=CVI (MI$)**	A string (MI$) created with the MKI$ function is converted back to the original integer value and assigned to A%.
	B%=CVI (MI$,S)	The contents of character string MI$ are re-arranged, and then the string is converted back to the original integer value. The value is assigned to B%.

DESCRIPTION

- The **CVI** function converts a character string created with the **MKI$** function back to the original integer value.
- Since all data written to random files must be character strings, the **MKI$** function is used to convert integer values to 2 byte character strings so they can be written to the file. When this data is read back from the file, it is necessary to convert it back to the original integer value. The **CVI** function is used for this purpose.
- The **CVI** function can convert only character strings that were converted from an integer value using the **MKI$** function. If any other character string is specified for **<character string expression>**, meaningful results will not be produced, or an **"Illegal function call"** error may occur.
- Data converted to a character string with the **MKI$** function is not only used for writing to a random file, it can also be used for data communication.
- If the **[,s]** option is specified, the character string to be converted is first re-arranged as shown below. Following the re-arrangement, it is converted into an integer value.



REMARK

See MKI$.

- Convert a character string created with the MKS$ function back to a single precision real number.

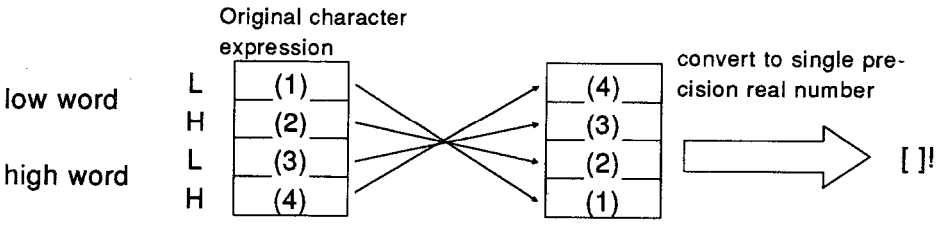
ConVert to Single

SYNTAX	**CVS (<character string expression> [,s])**

	character string expression	Specifies the string that was created with the MKS$ function.
EXAMPLE	**A ! =CVS (MS$)**	A string (MS$) created with the MKS$ function is converted back to the original single precision real number value and assigned to A!.
	B ! =CVS (MS$,S)	The contents of character string MS$ are re-arranged, and then the string is converted back to the original single precision real number value. The value is assigned to B!.

DESCRIPTION

- The **CVS** function converts a character string created with the **MKS$** function back to the original single precision real number.
- Since all data written to random files must be character strings, the **MKS$** function is used to convert single precision real number values to 4 byte character strings so they can be written to the file. When this data is read back from the file, it is necessary to convert it back to the original single precision real number value. The **CVS** function is used for this purpose.
- The **CVS** function can convert only character strings that were converted from a single precision real number value using the **MKS$** function. If any other character string is specified for **<character string expression>**, meaningful results will not be produced, or an **"Illegal function call"** error may occur.
- Data converted to a character string with the **MKS$** function is not only used for writing to a random file, it can also be used for data communication.
- If the **[,s]** option is specified, the character string to be converted is first re-arranged as shown below. Following the re-arrangement, it is converted into a single precision real number value.



REMARK

See MKS$.

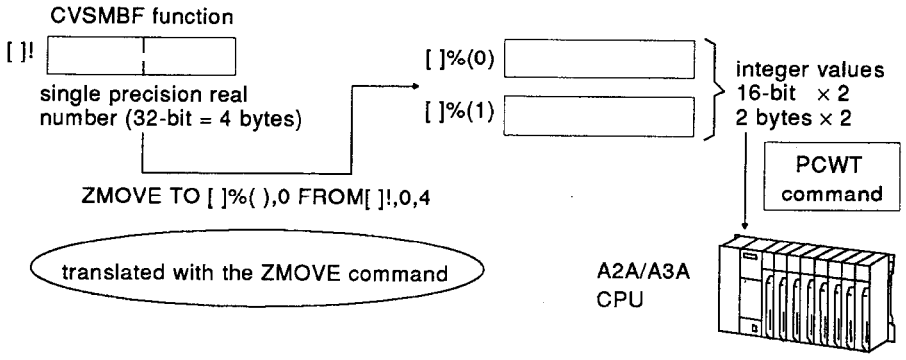
- Converts a string that was converted from an AD51H-BASIC single precision real number with the MKS$ function into the internal representation of floating point real numbers used by the A2A and A3A CPUs (IEEE format single precision internal representation).

Convert Single Microsoft Binary Format

SYNTAX	**CVSMBF (<character string expression>)**	
	character string expression	Specifies a string that was converted from a single precision real number with the MKS$ function.
EXAMPLE	**A!=CVSMBF(MKS$(123.45))**	Convert the single precision real number value 123.45 to the internal representation of floating point real numbers used by the A2A and A3A CPUs (IEEE format single precision internal representation) and assign it to A!.

DESCRIPTION

- The **CVSMBF** function converts a string that was converted from an AD51H-BASIC single precision real number with the **MKS$** function into the internal representation of floating point real numbers used by the A2A and A3A CPUs (IEEE format single precision internal representation).
- The AD51H-BASIC internal representation of real numbers, and the A2A and A3A CPUs internal representation of floating point real numbers (IEEE format) are different. The **CVSMBF** function is used to convert between the two representations.
- **<Character string argument>** should always be a string that was converted from a single precision real number with the **MKS$** function. Strings created with the **MKD$** or **MKI$** functions cannot be converted correctly.
- The value returned by **CVSMBF** is a single precision real number. Therefore, this value should be assigned only to single precision real type variables.
- When writing values that were converted with the **CVSMBF** function to the A2A and A3A CPUs with the **PCWT** command, the converted values must first be stored in an integer array as two values using the **ZMOVE** command.



- In addition to being used for the A2A and A3A CPUs, values converted with the **CVSMBF** function can be used when transferring data to a system which uses the IEEE format for internally representing single precision real numbers.

PROGRAMMING EXAMPLE

```
10  'Convert a single precision real number to an AnA CPU floating point real number and
    write it to D0, D1
20  DIM TBL%(5),A%(1)                          :'Define an array
30  A!=5.6767                                  :'Define a single precision real number
50  A$=MKS$(A!)                                :'Convert to a character string
70  B!=CVSMBF(A$)                              :'Convert to IEEE (AnA CPU) format
90  ZMOVE TO A%( ),0 FROM B!,0,4               :'Store to array A%
100  TBL%(0)=255                               :'Set the communication station number to
                                                the host
110  TBL%(1)=1                                 :'Select write to device memory
120  TBL%(2)=2                                 :'Select word units
130  TBL%(3)=18                                :'Specify the data register
140  TBL%(4)=0                                 :'Specify the device number
150  TBL%(5)=2                                 :'Specify the operation point count
160  PCWT TBL%( ),A%( )                        :'Execute the write operation
170  END
```

REMARKS

(1) The value returned by the CVSMBF function has no meaning within AD51H-BASIC.

(2) See CVDMBF, MKDMBF$, and MKSMBF$.

- Specifies the numeric values and character strings to be read with the READ command.

DATA

SYNTAX	**DATAΔ<constant>[<,constant> . . .]**	
	constant	Any desired numeric constant (fixed decimal constant, floating point constant, integer constant) or character string constant can be specified.
EXAMPLE	**DATA 1, 2, 3**	Specifies data (1,2,3) to be read with the READ command.

DESCRIPTION

- The **DATA** command specifies the data to be read with the READ command.
- The **DATA** command is a non-executed command. It can be written anywhere in the program.
- As many constants as can fit on one line (255 characters) can be set with one **DATA** command. Constants must each be separated with commas. Character string constants that include the comma, semicolon or colons characters, or that have whitespace at the beginning or end must be enclosed in quotes.
  Otherwise, quotes are not necessary.
- Several **DATA** commands can be included in a single program.
- The **READ** command reads data from the various DATA commands starting with the **DATA** command with the lowest line number and proceeding in order to the one with the highest line number.
- The constants in the **DATA** command and the corresponding variables in the **READ** command must have matching data types.
  The **RESTORE** command can be used to re-read constants from any specified **DATA** command.
- The **DATA** command is normally used together with the READ command, so it is advisable to refer to the **READ** command also.
- If the amount of data read by the **READ** command is greater than the available amount of data, an **"Out of DATA"** error occurs.

PROGRAMMING EXAMPLE

```
10  'Read data from the DATA command with the READ command
20  DATA 1,2,"ABC"
30  READ A,B,C$,D$,E                          :'Read the data
40  DATA "AD51H",90
50  PRINT "A=";A,"B=";B,"C$=";C$              :'Display the data
60  PRINT "D$=";D$,"E=";E
70  END

RUN
A= 1            B= 2            C$=ABC
D$=AD51H        E=90
OK
```

REMARK

See READ and RESTORE. Also see section 3.5.2.

- Sets the PC CPU year, month and day.
- Reads the year, month and day from the PC CPU.

DATE $

SYNTAX	**DATE$ = "<year>/<month>/<day>[/<day of the week>]"** **DATE$**

	year	A character string representing the year can be specified in the range "1990" to "2089".
	month	A character string representing the month can be specified in the range "01" to "12".
	day	A character string representing the day can be specified in the range "01" to "31".
	day of the week	A character string representing the day of the week can be specified.
EXAMPLE	**DATE$="1990/07/30/MON"**	The date is set to Monday, July 30, 1990.
	A$=DATE$	The date is read and assigned it to A$.

DESCRIPTION

- Sets the date of the PC CPU in which the AD51H is installed. Also can be used to read the date from the PC CPU.
- Before setting the date, the time should be set with the **TIME$** function Depending on the value that is set for the time, the date may become incorrect.
- The format for setting the date is as follows:

  DATE$ = "year/month/day[/day of week]"

  (1) The year, month, day and day of the week must be separated with a slash character.

  (2) The year is specified with four digits.

  (3) If the month or day are in the range 1 to 9, a zero must be added at the beginning (01 to 09).

  (4) The days of the week must be abbreviated as follows:

MON	**FRI**
TUE	**SAT**
WED	**SUN**
THU	

  (5) It is not possible to set only one part of the date (i.e., only the month,year, etc.).

POINTS

(1)The DATE$ function is only valid if the PC CPU has a clock/calendar device. For PC CPUs that have no such device (A[ ], A3H, A0J2(H)CPU),the DATE$ function cannot be executed.

(2)If the string which represents the year is not set in the range 1990 to 2089,normal operation is not possible.

PROGRAMMING EXAMPLE

```
10  'Set and read the calendar
30  A$="1991/04/01/MON"                    :'Set the year, month, day, and day of
                                            the week
40  DATE$=A$                               :'Write the year, month, day, and day of
                                            the week
60  B$=SPACE$(14)                          :'Create a blank variable
70  B$=DATE$                               :'Read the year, month, day, and day of
                                            the week
80  PRINT "Calendar setting-->";B$         :'Verify the contents that were written
90  END

RUN
Calendar setting-->1991/04/01/MON
OK
```

- Define all variables beginning with the characters in the specified range as double precision real variables.

DEFine DouBLe

SYNTAX	**DEFDBL Δ <character> [- <character>] [,<character> [-<character>], ···**

EXAMPLE	**DEFDBL A, C-E**	Define all variables that begin with A, C, D and E as double precision real number variables.

DESCRIPTION

- The **DEFDBL** command defines all variables beginning with the characters in the specified range as double precision real variables. All variable names beginning with **<character>** or all variable names beginning with characters in the range **<character - character>**, are defined as double precision real type variables.

- Variable type declaration characters have priority over the DEFDBL command, so if a variable includes a type declaration character, the variable's type corresponds to the type of the type declaration character even if it was declared with the DEFDBL command.

Example

```
10 DEFDBLA
20 A=10/3          double precision
30 A%=10/3         integer
40 A!=10/3         single precision
50 PRINT A,A%, A!
60 END
```

```
RUN
3.3333333333333333  3  3.3333
OK
■
```

- Variables that are declared as double precision real type with the **DEFDBL** command are identical to those which have the type definition character # following the variable name.

Example

```
10 DEFDBLA
20A=10/3...10/3=3.333...
                    stored to A
30 A#=5/3...5/3=1.666...
                    stored to A
40 PRINT A
50 END
```

```
RUN
  1.666666666666667...
OK
■
```

The variable A is specified stored to A in the PRINT command ,but the value of A# is displayed. There ,A and A# are the same variable.

- All variables that have not been defined with the **DEFDBL** command, and that do not have a type declaration character at the end of the variable name are considered single precision real type.

POINT

Whitespace may not be inserted between DEF and DBL.
The command must be entered as one continuous word.

REMARK

See DEFINT, DEFSNG and DEFSTR. Also see section 2.9.

- Defines a user function and assigns it a name.

DEFine FuNction

SYNTAX	**DEF∆FN<name> [(<formal argument> [, formal argument>], ...)] =<definition expression>**

name	Specifies the name to be assigned to the defined function.
formal argument	Specifies the variables to be used in the function definition expression.
function definition expression	Specifies the expression to return the value of the function.

EXAMPLE

DEF FNCOT (X)=1/TAN (X) Define the function fncot(x) to equal cot x (= 1/tan x).

DESCRIPTION

- The **DEFFN** command defines a user function and assigns it a name.
- To use functions that are not included in BASIC, it is necessary for the user to make them him/herself. Using the **DEF FN** command, the user can define his/her own functions.
- **<Name>** specifies the function when the FN command is used to call the function. Name must follow the same rules as variable names. If a type declaration character (%, !, #, $) is included in **<name>**, the corresponding function will return a value of the specified type. If a type declaration character is not included, the function will return a single precision real value.

  Example

  DEF FNA%(X,Y)=X+Y ——→ **Function FNA%(X,Y) will return an integer value.**

  DEF FNB#(X,Y)=X/Y ——→ **Function FNB#(X,Y) will return a double precision real number value.**

  DEF FNC$(X$)= "OK"+X$ ——→ **Function FNC%(X,Y) will return a character string value.**

- The **<formal arguments>** represent the variables used in the function definition expression. When the function is called, the formal arguments are passed into the corresponding variables **<real arguments>**, and the function expression is calculated. The **<formal arguments>** may be omitted (a function with no arguments can be defined).
- The function definition expression defines the function's method of calculation. The **DEF FN** statement can be up to one line (255 characters) long.

- The variables included in the function definition expression are used only to define the manner of calculation, and do not have any relation to the same variables used within the program; their values cannot be changed or referenced. Also, it is not necessary to include a **<formal argument>** for all variables used in the function definition expression. When a variable has no **<formal argument>**, its current value is used.

  DEF FNA (X, Y)=A*X+B*Y

- User defined functions are called using the **FN** command.
- The **DEF FN** command cannot be used in direct mode.

PROGRAMMING EXAMPLE

```
10 'Define the equation that calculates the area of a triangle to function FNA
20 DEF FNA(A,B)=(A*B)/2                    :'(base * height)/2
30 INPUT "BASE =";C                        :'Input the base
40 INPUT "HEIGHT =";D                      :'Input the height
50 E=FNA(C,D)                              :'Calculate
60 PRINT "Area of triangle =";E            :'Display results of calculation
70 END

RUN
BASE =? 10
HEIGHT =? 2
Area of triangle = 10
OK
```

- Define all variables beginning with the characters in the specified range as integer variables.

DEFine INTeger

SYNTAX	**DEFINTΔ<character> [- <character>] [,<character> [-<character>]]...**
EXAMPLE	**DEFINT B,F-G** — Define all variables that begin with B, F and G as integer variables.

DESCRIPTION

- The **DEFINT** command defines all variables beginning with the characters in the specified range as integer variables. All variable names beginning with **<character>** or all variable names beginning with characters in the range **<character - character>**, are defined as integer type variables.

- Variable type declaration characters have priority over the DEFINT command, so if a variable includes a type declaration character, the variable's type corresponds to the type of the type declaration character even if it was declared with the DEFINT command.

Example

```
10 DEFINT A
20 A=10/3          (integer)
30 A#=10/3         (double precision)
40 A!=10/3         (single precision)
50 PRINT A, A#, A!
60 END
```

```
RUN
3  3.333333333333333  3.3333
OK
■
```

- Variables that are declared as integer type with the **DEFINT** command are identical to those which have the type definition character % following the variable name.

Example

```
10 DEFINT A
20 A=100
30 A%=777
40 PRINT A
50 END
```

```
RUN
    777
OK
■
```

... The variable A is specified in the PRINT command, but the value of A% from line 30 is displayed. Therefore, A and A% are the same variable.

- All variables that have not been defined with the **DEFINT** command, and that do not have a type declaration character at the end of the variable name are considered single precision real type.

POINT

Whitespace may not be inserted between DEF and INT. The command must be entered as one continuous word.

REMARK

See DEFDBL, DEFSNG and DEFSTR. Also see section (3).

- Define all variables beginning with the characters in the specified range as single precision real variables.

DEFine SiNGle

SYNTAX	**DEFSNGΔ<character> [- <character>] [, <character> [- <character>]],...**

EXAMPLE	**DEFSNG C, X-Z**	Define all variables that begin with C, X, Y and Z as single precision real number variables.

DESCRIPTION

- The **DEFSNG** command defines all variables beginning with the characters in the specified range as single precision real variables. All variable names beginning with **<character>** or all variable names beginning with characters in the range **<character - character>**, are defined as single precision real type variables.

- Variable type declaration characters have priority over the DEFSNG command, so if a variable includes a type declaration character, the variable's type corresponds to the type of the type declaration character even if it was declared with the DEFSNG command.

Example

```
10 DEFSNG A
20 A=10/3          (single precision)
30 A#=10/3         (integer)
40 A!=10/3         (double precision)
50 PRINT A, A%, A#
60 END
```

```
RUN
3.3333    3  3.3333333333333333
OK
■
```

- Variables that are declared as single precision real type with the DEFSNG command are identical to those which have the type definition character ! following the variable name.

Example

```
10 DEFSNG A
20 A=10/3£10/3=3.333£stored to A
30 A!=5/3£5/3=1.666£stored to A!
40 PRINT A
50 END
```

```
RUN
  1.6666
OK
■
```

... The variable A is specified in the PRINT command, but the value of A! from line 30is displayed. Therefore, A and A! are the same variable.

POINT

Whitespace may not be inserted between DEF and SNG. The command must be entered as one continuous word.

REMARK

See DEFDBL, DEFINT and DEFSTR. Also see section 2.9.

- Define all variables beginning with the characters in the specified range as character string variables.

DEFine STRing

SYNTAX	**DEFSTRΔ<character> [- <character>] [, <character> [- <character>]], ...**
EXAMPLE	**DEFSTR M, P-R** — Define all variables that begin with M, P, Q and R as character string variables.

DESCRIPTION

- The **DEFSTR** command defines all variables beginning with the characters in the specified range as character string variables. All variable names beginning with **<character>** or all variable names beginning with characters in the range **<character - character>**, are defined as character string type variables.
- Variable type declaration characters have priority over the DEFSTR command, so if a variable includes a type declaration character, the variable's type corresponds to the type of the type declaration character even if it was declared with the DEFSTR command.

Example

```
10 DEFSTR A
20 A="10/3"        (character string)
30 A%=10/3         (integer)
40 A!=10/3         (single precision)
50 PRINT A, A%, A!
60 END
```

```
RUN
10/3        3     3.3333
OK
■
```

- Variables that are declared as character string type with the **DEFSTR** command are identical to those which have the type definition character $ following the variable name.

Example

```
10 DEFSTR A
20 A="Palo Alto"
30 A$="Cambridge"
40 PRINT A
50 END
```

```
RUN
 Cambridge
OK
■
```

... The variable A is specified in the PRINT command, but the value of A$ from line 30 is displayed. Therefore, A and A$ are the same variable.

- All variables that have not been defined with the **DEFSTR** command, and that do not have a type declaration character at the end of the variable name are considered single precision real type.

POINT

Whitespace may not be inserted between DEF and STR. The command must be entered as one continuous word.

REMARK

See DEFDBL, DEFINT and DEFSNG. Also see section 2.9.

- Defines an event which is used to synchronize execution between programs.
- Defines an event with an AD51H internal bit device (EM).

DEFine Z EVENT

SYNTAX	**DEF ZEVENTΔ<event number>** **DEF ZEVENTΔ<event number>, <device>**

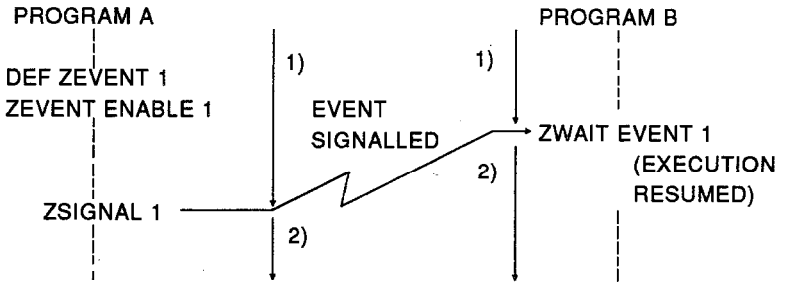
	event number	Specifies the number which indicates the event to be defined.
	device	Selects an AD51H internal bit device (EM) that will cause the event to be signalled when it is set ON.
EXAMPLE	**DEF ZEVENT 1**	An event which will be signalled with the ZSIGNAL command in order to synchronize execution between programs is defined as event number 1.
	DEF ZEVENT 3, "EM0001"	An event which will be signalled when internal bit device EM0001is set ON is defined as event number 3.

DESCRIPTION

- The **DEF ZEVENT** command defines all kinds of events that are used during multitasking.
- An event is a signal which allows one BASIC program to control another BASIC program.

DEFINING AN EVENT TO BE SIGNALLED WITH THE ZSIGNAL COMMAND

- An event is defined in order to synchronize execution with another program. In this case, only **<event number>** is specified. **<device>** should not be specified.



1) and 2) represent each program's range of execution during the same time period when the programs are running simultaneously.

- Events to be signalled with the **ZSIGNAL** command can only be signalled with the **ZSIGNAL** command. Such events are first defined with the **DEF ZEVENT** command, and then event signalling is enabled for the event with the **ZEVENT** command. Any time after this, if the **ZSIGNAL** command is executed, execution of any BASIC program that is waiting for that event's signal is resumed.

- The defined number is used to specify which event is to be enabled/disabled, signalled, or waited for. All BASIC programs handle the same **<event numbers>**.
  The value specified for number must be in the range 0 to 63. Events that are signalled by an AD51H internal bit device use the same **<event numbers>**.

- If an event number that has already been defined is specified in the DEF ZEVENT command, the contents of the previous definition become invalid, and the new definition becomes valid.

DEFINING EVENTS FOR USE WITH AD51H INTERNAL BIT DEVICES

- An event which will be signalled automatically when the condition of a specified AD51H internal bit device changes from OFF to ON is defined. In this case, both **<event number>** and **<device>** must be specified.

- Events defined using this procedure may be signalled either with the **ZSIGNAL** command, or when the bit device changes from OFF to ON. If this event is enabled with the **ZEVENT** command, when it is signalled by the **ZSIGNAL** command or by the bit device, execution of all programs waiting for the event is resumed.

- The defined **<event number>** is used to specify which event is to be enabled/disabled,signalled, or waited for. All BASIC programs handle the same event numbers.
  The value specified for **<event number>** must be in the range 0 to 15. Events that are used to synchronize program execution use the same event numbers.

POINT

Event numbers to be assigned to events which use an AD51H internal bit device must be specified in the range 0 to 15. If numbers in the range 16 to 63 are specified, an error occurs.

- **<Device>** specifies the AD51H internal bit device which will cause the event to be signalled when it is changed from OFF to ON. It should be specified as follows:

"E M [ ] [ ] [ ] [ ]"

'EM' must be used.

The internal bit device number is specified in the range 0 to 1023. When the device number specified is in the range 0 to 999, zeroes must be added to the number to make it four digits long.

EM1 ⟶ "EM0001"

EM260 ⟶ "EM0260"

POINT

If event processing is used, when the AD51H system is stopped or set up, it is necessary to disable all events to be handled in order to initialize event information managed by the system. Either of the following methods is possible:

1) Turn the power off and on again, or use the RESET switch.

2) Re-define all handled events either in the first BASIC program that is executed during AD51H system setup, or in the last BASIC program to finish execution.

Re-definition is performed as follows:

```
FOR 1%=0 TO 63
DEF ZEVENT 1%
NEXT 1%
```

REMARKS

(1) For more information on event processing, see section 8.2.

(1) See ZEVENT, ZSIGNAL and ZWAIT EVENT.

- Delete a specified range of a program.

DELETE

SYNTAX	DELETEΔ[[<line number 1>] -] [<line number 2>]

	line number 1	Specifies the first line number to be deleted.
	line number 2	Specifies the last line number to be deleted.
EXAMPLE	**DELETE 100 - 150**	All program lines form line 100 to 150 are deleted.
	DELETE 50 -	All program lines from line 50 to the end of the program are deleted.
	DELETE - 200	All program lines from the beginning of the program to line 200 are deleted.
	DELETE 70	Program line 70 is deleted.

DESCRIPTION

- The **DELETE** command deletes the range of program lines specified with **<line number 1>** and **<line number 2>**.
- If both **<line number 1>** and **<line number 2>** are specified, all program lines in between the two specified lines are deleted (inclusive).
- If **<line number 1>** is specified and followed by a hyphen, all program lines from the specified line to the end of the program are deleted.
- If **<line number 2>** is specified and preceeded with a hyphen, all program lines from the beginning of the program to the specified line are deleted.
- If only **<line number 1>** is specified, only the specified line is deleted.
- If the **DELETE** command is executed within a program, the following may occur:
  IN EDIT MODE
  BASIC enters the "waiting for command" condition.
  IN EXECUTE MODE
  The "not executing" condition becomes active.
- If the specified **<line number>** does not exist, an **"Illegal function call"** error occurs.

REMARK

See CHAIN. Also see section 3.2.3.

- The size of an array variable is specified, an only the necessary memory area is allocated to the array.

DIMension

SYNTAX	**DIMΔarray name (numeric expression [,numeric expression] ···)** **[, array name (numeric expression [,numeric expression] ···)]**

	array name	Specifies the name to be used with the array variable.
	numeric expression	Specifies the size of the array.
EXAMPLE	**DIM A(2,2)**	Indicates a two dimensional numeric value array.

A(0,0)	A(1,0)	A(2,0)
A(0,1)	A(1,1)	A(2,1)
A(0,2)	A(1,2)	A(2,2)

DESCRIPTION

- The **DIM** command specifies the size of an array variable, and allocates only the necessary memory area to the array.
- The following kinds of arrays may be defined:

  A(n) ..... numeric value array

  A%(n) .....integer array

  A!(n) ..... single precision array

  A#(n) ..... double precision array

  A$(n) ..... character string array

- **<Numeric expressin>** can be specified as large as 32767. However, if too much memory is reserved, memory may not be sufficient.
- When using an array, if the value of the subscript is larger than the **<numeric expression>** specified in the **DIM** command for the variable, a **"Subscript out of range"** error occurs.
- **<Numeric argument>** cannot be less than 0.
- If an array name is used that has not been defined with the **DIM** command, its highest subscript is assumed to be 10.

POINT

Some BASIC commands cannot use arrays that have been "implicitly defined" as described above.
Therefore, it is suggested that all arrays be explicitly defined with the DIM command before they are used.

- When a numeric array is specified in the **DIM** command, all elements are initialized to 0, if a character array is specified, all elements are initialized to the empty string.
- If an array variable that has been set up with the **DIM** command is redefined with the **DIM** command, a **"Redimensioned array"** error occurs.
  To redefine an array, first delete its definition with the **ERASE** command, and then redefine it with the DIM command.

REMARK

See ERASE. Also see section 3.8.

For information on defining the array type, refer also to DEFINT, DEFSNG, and DEFDBL.

- In edit mode, program execution is completed and the "waiting for input" condition becomes active .
- In execute mode, program execution is completed and the "not executing" condition becomes active.

END

SYNTAX	**END**	
EXAMPLE	**END**	Program execution is ended, all open files are closed, and the "waiting for command input" condition becomes active. (edit mode) Program execution is ended, all open files are closed, and the "not executing" condition becomes active. (execute mode)

DESCRIPTION

- Program execution is halted, and the following events occur:

EDIT MODE

- All open files are closed, and the "waiting for command input" condition becomes active.
- Unlike the **STOP** command, a "Break in " message is not displayed.
- At the end of the program, the END command can be omitted. However, if the **END** command is omitted, files are not closed.

EXECUTE MODE

- Program execution is ended, and the "not executing" condition becomes active.
  Programs in the "not executing" condition may be re-executed with the **ZSTART** command.

REMARK

See CLOSE, RUN and STOP.

EOF function

- Returns the value -1 (true) when the end of a sequential file is detected.

End Of File

SYNTAX	**EOF (<file number>)**	
	file number	Specifies the file number assigned to the file with the OPEN command.
EXAMPLE	**IF EOF(1) THEN CLOSE #1**	If the end of sequential file #1 is detected(EOF function returns TRUE), the file is closed.

DESCRIPTION

- The **EOF** function detects the end of a sequential file specified by **<file number>**.
- If the end of the file is detected, the function returns the value TRUE (-1), otherwise, it returns FALSE (0).

REMARK

See LOC, LOF and OPEN. Also see chapter 6.

- Deletes from memory an array that was defined with the DIM command.

ERASE

SYNTAX	**ERASE∆<array variable >[<, array variable >....]**	
	array variable	Specifies an array variable that was defined with the DIM command.
EXAMPLE	**ERASE A, B#**	Array variables A and B$ are deleted.

DESCRIPTION

- The **ERASE** command deletes from memory an array that was defined with the **DIM** command.
- An array that has been deleted with the **ERASE** command can be redefined with the **DIM** command.
- All data that is stored in an array is lost when the array is deleted with the **ERASE** command.
- If an array is redefined with the **DIM** command without first being deleted with the **ERASE** command, a **"Redimensioned array"** error occurs.

PROGRAMMING EXAMPLE

```
1   'Delete an array with the ERASE command and redefine an array with the same name.
10  DIM A(99)
20  DIM A(199)

RUN
Redimensioned array in 20
OK
15 ERASE A
LIST
1 'Delete an array with the ERASE command and redefine an array with the same name.
10 DIM A(99)
15 ERASE A
20 DIM A(199)
OK
RUN
OK
```

REMARK

See DIM and CLEAR.

- Returns the line number where an error was detected.

ERror Line

SYNTAX	**ERL**	
EXAMPLE	**E = ERL**	The line number where the error was detected is stored in E.

DESCRIPTION

- When an error occurs, the **ERL** function returns the line number of the line where the error was detected.
- The **ERL** function is used within the error management routine to branch to a different procedure depending on the line number where the error was detected.
- If the command which caused the error was executed in direct mode, the value of **ERL** is 65535.
- Since the **ERL** function has a fixed purpose, it cannot be used on the left side of the equal sign in a **LET** command or similar usages.

PROGRAMMING EXAMPLE

```
10   'Display the line number where an error occurred.
20   ON ERROR GOTO 100                        :'Branch to line 100 if an error occurs
50   ERROR 10                                 :'<--this line causes an error
60   END
100   PRINT "ERROR LINE =";ERL                :'Display the line number where the error
                                               occurred
110   RESUME 60

RUN
ERROR LINE = 50
OK
```

REMARK

See ON ERROR GOTO, RESUME and ERR. Also see chapter 9.

- Returns the error code of a detected error.

ERRor code

SYNTAX	**ERR**	
EXAMPLE	**E = ERR**	The error code detected for an error is stored in E.

DESCRIPTION

- When an error occurs, the **ERR** function returns the error code for the error that was detected.
- The **ERR** function is used within the error management routine to branch to a different procedure depending on the contents of the error detected.
- Since the **ERR** function has a fixed purpose, it cannot be used on the left side of the equal sign in a **LET** command or similar usages.
- For information on error codes, see Appendix 4.4.

PROGRAMMING EXAMPLE

```
10  'Display an error code
20  ON ERROR GOTO 100                    :'Branch to line 100 if an error occurs
50  ERROR 10                             :'<--this line causes error 10
60  END
100  PRINT "ERROR CODE =";ERR            :'Display the error code
110  RESUME 60

RUN
ERROR CODE = 10
OK
```

REMARK

See ON ERROR GOTO, RESUME and ERL. Also see chapter 9.

- Causes an error with the specified error code to occur.

ERROR

SYNTAX	**ERROR∆<integer expression>**	
	integer expression	Specifies the error code of the error to be signalled.
EXAMPLE	**E = ERROR 2**	Error number 2 (syntax error) is signalled.

DESCRIPTION

- The **ERROR** command causes an error with the specified error code to occur.
- If <integer expression> is a value which corresponds to a BASIC error code (see Appendix 4.4), the **ERROR** command causes the associated error to occur.
- If an error code is specified which corresponds to an error that has no error message, the error **"Unprintable error"** is signalled.
- The **ERROR** command is used to verify the operation of the error management routine.

REMARK

See ON ERROR GOTO, ERR and ERL. Also see chapter 9.

- Returns the value of the exponential function with base e (e = 2.718281).

EXPonential

SYNTAX	**EXP (<numeric expression>)**	
	numeric expression	Specifies the power n in e^n.
EXAMPLE	**E = EXP(10)**	EXP(10) is calculated and the result is stored in E.

DESCRIPTION

- The **EXP** function returns the value of the exponential function with base **e** (e = 2 .718281).
- The value specified for **<numeric expression>** must be less than 87.33655. If this value is exceeded, an **"Overflow"** error occurs.
- The **EXP** function is calculated in single precision.

PROGRAMMING EXAMPLE

```
10  'Calculate the value of e^5.2.
20  A=5.2
30  B=EXP(A)                                  :'Calculate e^5.2
50  PRINT "e^5.2=";B
60  END

RUN
e^5.2= 181.272
OK
```

REMARK

See LOG.

- Assigns the specified variable an area within a random file.

FIELD

SYNTAX	**FIELDΔ<#>, <file nember>, <field length> AS <character string variable> [,<field length> AS <character string variable>] . . .**

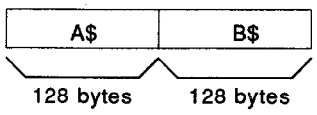
file number	Specifies the file number of the random file.
field length	Specifies the number of characters to be assigned within the buffer.
character string variable	Specifies the character string variable for the part specified in field length.

EXAMPLE

FIELD #1, 128 AS A$, 128 AS B$

The variables A$ and B$ are defined to be used in the program for the file buffer of random file number 1, and each variable is assingned 128 bytes.

random file buffer



DESCRIPTION

- The **FIELD** command assigns a portion of the file buffer of the random file specified by **<file number>** to variables that are used within the program. Reading from and writing to the file buffer can then be accomplished in the program via the specified variables.
- The **FIELD** command must be executed before data can be read with the **GET** command or written with the **PUT** command.
- The **LSET** and **RSET** commands should be used for writing to the random file buffer.
- The total number of bytes reserved by a single **FIELD** command (the total of all **<field length>** arguments) cannot exceed the total length of the file buffer,which is 256 bytes. If the total number of bytes reserved exceeds 256, a **"Field overflow"** error occurs.

REMARKS

(1) It is possible to execute several FIELD commands with the same file number. Each FIELD command assigns the variables to the buffer starting at the head of the buffer, so all assignments are simultaneously active.

(2) See OPEN. Also see chapter 6.

- Displays the file names of the files stored on a memory card, floppy disk or hard disk.

FILES

SYNTAX	**FILES [" [<drive number>:] [<system name>\] [<file name] "] [,S]**

	drive number	Specifies the memory card or disk whose files are to be displayed.
	system name	Specifies the system name where the files to be displayed are stored.
	file name	Specifies the name of the file to be displayed.
EXAMPLE	**FILES "0:", S**	The name, size, and date and time of creation of all files with no system name on the memory card inserted in memory card slot [1] are displayed.
	FILES "2:TEST\"	The names of all files stored in system name "TEST" on the IBM PC/AT floppy drive are displayed.

DESCRIPTION

- The **FILES** command displays the names of all files in the specified **<system name>** on the memory card, floppy disk or hard disk specified by **<drive number>**.
- If **<file name>** is specified, only the specified file is displayed.
  The following table shows the **<drive number>** specifications and their corresponding storage device.

  EDIT MODE, EXECUTE MODE

  AD51H Memory Card [1] 0

  AD51H Memory Card [2] 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFFLINE PROGRAMMING

  PC/AT FDD 1

  PC/AT HDD[C] 3
- If the **[,S]** option is specified, the file name, size and date and time of creation are displayed.
- If **<system name>** is omitted, all files with no system name are displayed.
- Wildcards may be used in the **<file name>** specification. For information on how to use wildcards, see appendix 1.2.

REMARK

See LFILES. Also see section 3.4.1.

- Truncates the decimal portion of a numeric value, and returns only the integer portion.

FIX

SYNTAX	**FIX (<numeric expression>)**	
	numeric expression	Specifies the numeric value which is to be truncated at the decimal point.
EXAMPLE	**FIX(1.28)→1**	The decimal portion of the value 1.28 is truncated and the value 1 is returned.
	FIX(-1.28)→-1	The decimal portion of the value -1.28 is truncated and the value -1 is returned.

DESCRIPTION

- The **FIX** function truncates the decimal portion of a numeric value, and returns only the integer portion.
- The **FIX** function differs from the **INT** function in that the FIX function truncates negative numbers as well as positive numbers, while the INT function rounds up negative numbers.
- The expression **FIX (X)** is equivalent to the expression **(SGN(X))**× **(INT (ABS(X)))**

EXAMPLE:**X=-3.54**

(SGN(-3.54)) × (INT (ABS(-3.54)))

↓

INT (Absolute value 3.54)

↓

(Sign-1) × (truncate the decimal portion 3) =-3

PROGRAMMING EXAMPLE

```
10  'Truncate the decimal portion of numeric values.
20  A=3.95                                  :'Define numeric values
30  B=-3.95
40  C=FIX(A!)                               :'The decimal portion of A is truncated and
                                             the remaining value is stored in C
50  D=FIX(B!)                               :'The decimal portion of B is truncated and
                                             the remaining value is stored in D
60  PRINT       A=";A,"     B=";B
70  PRINT "FIX(A)=";C,"FIX(B)=";D           :'The truncated values
80  END

RUN
     A= 3.95     B=-3.95
FIX(A)= 3   FIX(B)=-3
OK
```

REMARK

See CINT, INT. Also see sections 2.7.1 and 2.8.1.

- Repeat execution of a sequence of commands a specified number of times.

FOR~NEXT

SYNTAX	**FORΔ<variable name>=<inital value>ΔTO<end value>Δ[STEP<increment>]** **NEXT [<variable name>] [,<variable name>]**

	variable name	Specifies the variable to control the loop.
	initial value	Specifies the starting value of the variable that controls the loop.
	end value	Specifies the ending value of the variable that controls the loop.
	increment	Specifies the increment by which the variable that controls the loop changes.
EXAMPLE	**FOR 1=0 TO 100 STEP 2** ⋮ **NEXT 1**	The variable I is increased from 1 to 100 by increments of 2 while the procedure between the FOR and NEXT commands is repeated.

DESCRIPTION

- The **FOR~NEXT** command repeats execution of all commands between the FOR and the NEXT a specified number of times.
- Only integer or single precision real type variables may be specified for **<variable name>**. Character string and double precision real type variables cannot be used.
- The commands in between the **FOR** command and the **NEXT** command are executed, and the specified increment is added to the value of the variable. The variable is then compared to the end value.
  If the value of the variable is less than or equal to the end value, execution is returned to the command following the **FOR** command.
  If the value of the variable is more than the end value, the command following the **NEXT** command is executed.
- If the **STEP** is omitted, the variable increment is assumed to be 1.
  Also, if the value specified for the **STEP** is negative, the end value must be less than the initial value. In this case, after each execution of the loop, the value of the variable is decreased, and the loop ends when the value of the variable is less than the end value.
  If the **STEP** value is positive, and the initial value is larger than the end value, or if the **STEP** value is negative, and the initial value is smaller than the end value, the commands between the **FOR** command and the **NEXT** command are executed only once.

- **FOR~NEXT** loops may be nested; a **FOR~NEXT** loop can be placed inside of another **FOR~NEXT** loop.
  If loops are nested, the variables used by each loop must be different.
  Also, one **FOR~NEXT** loop must be placed inside of another.

```
FOR I=0 TO 10
      :
FOR J=0 TO 10
      :
NEXT J
      :
NEXT I
```

repeat repeat

When nested loops have the same ending conditions, several variable names may be included in one **NEXT** command.
However, the variable names must be written in order based on whichever **FOR** command is closest.

```
FOR I=0 TO 10
      :
FOR J=0 TO 10
      :
NEXT J,I
```

- If a **NEXT** command exists with no corresponding **FOR** command, a **"NEXT without FOR"** error occurs.

PROGRAMMING EXAMPLE

```
10  'Make a nine by nine table
30  FOR I=1 TO 9                          :'Loop with I from 1 to 9
40   FOR J=1 TO 9                         :'Loop with J from 1 to 9
50    PRINT USING "####";I*J;             :'Display the value of I*J
60   NEXT J
70   PRINT                                :'End the line
80  NEXT I
90  END

RUN
    1   2   3   4   5   6   7   8   9
    2   4   6   8  10  12  14  16  18
    3   6   9  12  15  18  21  24  27
    4   8  12  16  20  24  28  32  36
    5  10  15  20  25  30  35  40  45
    6  12  18  24  30  36  42  48  54
    7  14  21  28  35  42  49  56  63
    8  16  24  32  40  48  56  64  72
    9  18  27  36  45  54  63  72  81
OK
```

REMARK

See ERASE and WHILE ~ WEND. Also see section 3.6.3.

- Initializes the file area of a memory card (performs a logical format).

FORMAT

SYNTAX	**FORMATΔ<drive number>**	
	drive number	Specifies the memory card to be initialized (logically formatted).
EXAMPLE	**FORMAT 0**	The memory card inserted in memory card slot [1] is initialized (logically formatted).

DESCRIPTION

- The **FORMAT** command initializes (logically formats) the file area of a memory card so that BASIC program and data files can be written to it.
- If the file area of a memory card has not been initialized (logically formatted) with the **FORMAT** command, BASIC program and data files cannot be written to the memory card.
- Simply initializing (logically formatting) a memory card with the BASIC **FORMAT** command does not make the memory card usable. To make a memory card usable, it must first be completely initialized (physically formatted) in file maintenance mode or in system mode. Following the complete initialization (physical format) the BASIC **FORMAT** command is executed to initialize (logically format) the file area.

  MANUALS FOR REFERENCE

  Complete initialization (physical format) of a memory cardin file maintenance mode. ..... SW[ ]IX-AD51HPE: AD51H Software Package Operating Manual

  Complete initialization (physical format) of a memory card in system mode. ..... AD51H/AD51H-S3 User's Manua

- The **<drive number>** specifies which memory card's file area is to be initialized (logically formatted).
  The following shows the valid settings:

  Initialize the file area of the card inserted in Memory Card Slot [1] 0

  Initialize the file area of the card inserted in Memory Card Slot [2] 1

POINT

When the file area of a memory card is initialized with the FORMAT command, all program and data files stored on the memory card are lost. There is no way to retrieve lost data. Before using the FORMAT command, be sure to thoroughly verify the contents of the memory card.

REMARK

Refer to the SW[ ]IX-AD51HPE Software Package Operating Manual, and the AD51H/AD51H-S3 User's Manual.

- Returns the size of the remaining memory area in bytes.

FREe

SYNTAX	**FRE (<numeric expression>)** **FRE (<character string expression>)**

	numeric expression	A dummy numeric value constant or variable is specified.
	character string expression	A dummy character string constant or variable is specified.
EXAMPLE	**PRINT FRE (A)**	The number of bytes of memory remaining that can be used by BASIC programs is displayed.
	PRINT FRE ("TEST")	The number of bytes of memory remaining that can be used for character string management is displayed.

DESCRIPTION

- If **<numeric expression>** is specified in the **FRE** function, the size of the remaining memory area in the BASIC program area is returned in bytes.
  The remaining memory area may be used for the program, character string management, or variable storage.
  <Numeric expression> is a dummy value and has no meaning.
- If **<character string expression>** is specified, the size of the remaining memory area that can be used for character string management is returned in bytes.
  <Character string expression> is a dummy value and has no meaning.
- The value returned by the **FRE** function does not represent the total memory used by multitasking, but only the memory used in the task area where the **FRE** function was executed.
- For information on the memory map, see the AD51H/AD51H-S3 User's Manual.

PROGRAMMING EXAMPLE

```
10 'Display the size of the remaining memory area
20 PRINT "Task number area --> ";ZBAS          :'Display the task number area
30 PRINT "Remaining memory area=
   ";FRE(0);"bytes"                            :'Display the size of the remaining
                                                memory area
50 END

RUN
Task number area -->1
Remaining memory area= 4601 bytes
OK
```

REMARK

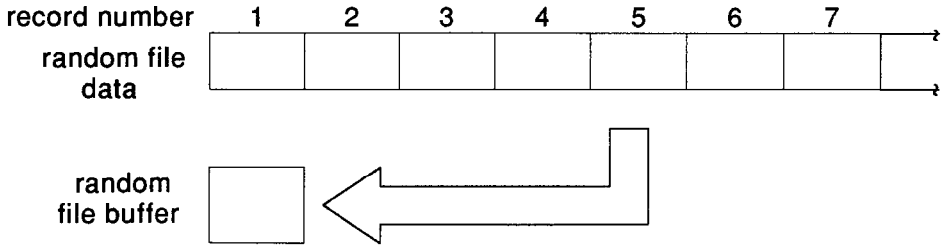
See CLEAR and ERASE.

- Read one record from a random file into the file buffer.

GET

SYNTAX	GETΔ[#] file number [,record number]

	file number	Specifies the file number assigned to the file when it was opened.
	record number	Specifies the number of the record to read from the random file.
EXAMPLE	**GET #1, 5**	Record number 5 from file number 1 is read into the random file buffer.



DESCRIPTION

- The **GET** command reads one record of data from a random file into the random file buffer.
- **<File number>** specifies the number that was used to open the random file from which the data will be read.
- **<Record number>** specifies the number of the record to read from the random file.
- If the **<record number>** is omitted, the record following the record specified in the most recently executed **GET** or **PUT** command is assumed.

  If the **<record number>** is omitted when the **GET** command is executed immediately following the **OPEN** command, record number 1 is assumed.

REMARK

See OPEN, FIELD, PUT and CLOSE. Also see chapter 6 .

- Reads data from the AD51H buffer memory, common memory or an internal device (ED).

GET MEMory

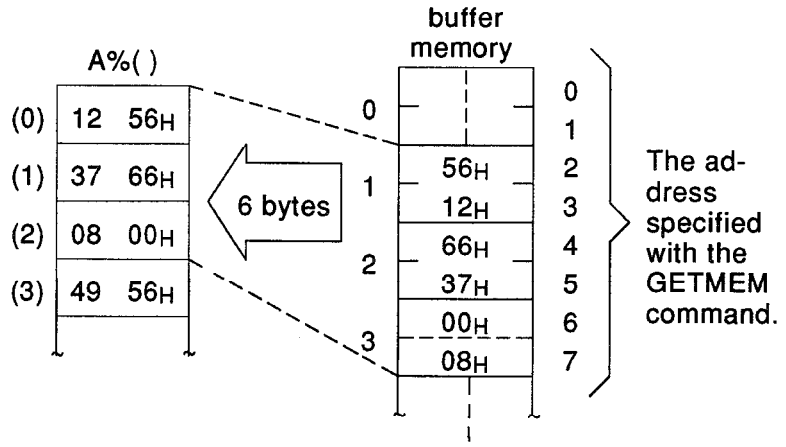
SYNTAX	**GETMEMΔTOΔ<destination storage area>, <offset 1>ΔFROMΔ<read source>Δ, <offset 2>, <number of bytes>**

read source	Specifies the buffer memory or common memory head address, or the internal word device head device.
destination storage area	Specifies the integer variable, integer array, character string variable or character string array in which to store the data that is read.
offset 1	Specifies the offset (in byte units) from the head of the destination storage area.
offset 2	Specifies the offset (in byte units) from the head of the read source.
number of bytes	Specifies the amount of data to read in byte units.

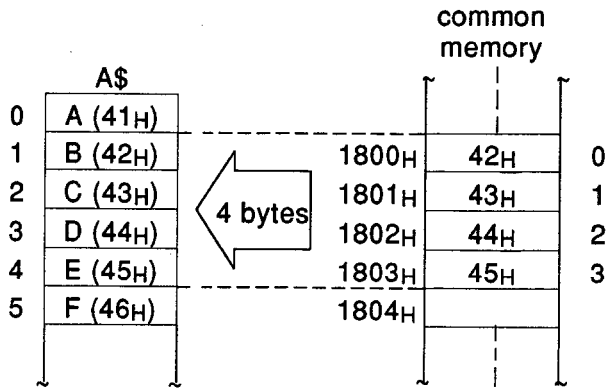
EXAMPLE

GETMEM TO A%( ), 0 FROM 0, 2, 6

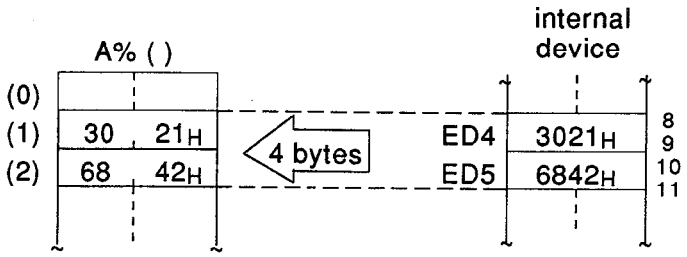
Six bytes of data from buffer memory 1 to 3 are read into A%(0) to A%(2).



GETMEM TO A$, 1 FROM &H1800, 0, 4 Four bytes of data are read from AD51H common memory (addresses &H1800 &H1803) into the second to fifth characters of A$.



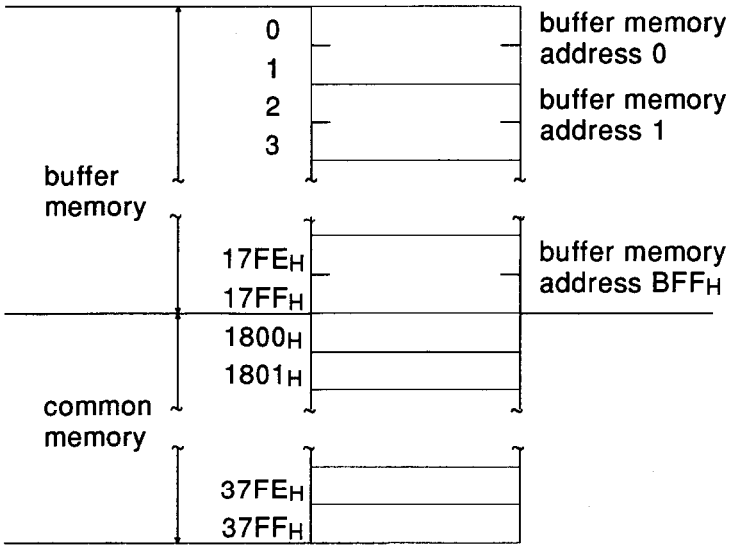
GETMEM TO A%( ), 2 FROM W@) (ED, 2), 4, 4
Four bytes of data are read from ED2 and ED3 into A%(1) and A%(2).



DESCRIPTION

- The **GETMEM** command reads the amount of data specified by **<number of bytes>** from the buffer memory, common memory or internal device (ED) area specified by **<read source>** and **<offset 2>** into the variable specified by **<destination storage area>** and **<offset 1>**.
- A buffer memory or common memory address, or an internal word device may be specified for **<read source>**.
  Buffer memory and common memory addresses are organized as follows.
  The head address to read from is specified with **<read source>**.

Specifying addresses with GETMEM



POINT

As shown above, one buffer memory address uses 2 bytes. Therefore buffer memory addresses should be specified the following way with the GETMEM command.

(buffer memory address specified with the GETMEM command) = (AD51H buffer memory address) × 2

When reading from an internal word device, the head device is specified the same as when using special variable W@ (ED,n).

To use ED 100 as the head device → **W@ (ED, 100)**

To use ED 20 as the head device → **W@ (ED, 20)**

- The variable in which to store the data that is read with the GETMEM command is specified with **<destination storage area>**.
  Before executing the **GETMEM** command, the variable that is to be used for the **<destination storage area>** must have dummy data stored in it. The amount of dummy data must be at least the **<number of bytes>** specified in the **GETMEM** command. If dummy data is not stored in the variable, when the **GETMEM** command is executed,an error occurs.

  integer variable **[ ]% = 0**

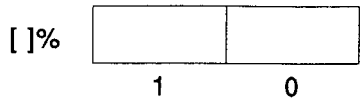
  character variable **[ ]$ = SPACE (255)**

- Even if the array variable to be used as the **<destination storage area>** uses less than ten elements, it must be defined with the **DIM** command. If the array is not defined with the **DIM** command, when the **GETMEM** command is executed, only the amount of read-in data equal to the array size as defined with the **DIM** command is stored.
  When using a character string array variable, dummy data should be stored as follows. The amount of dummy data must be at least the of bytes specified in the **GETMEM** command.

  [ ]$(n) = SPACE (255)

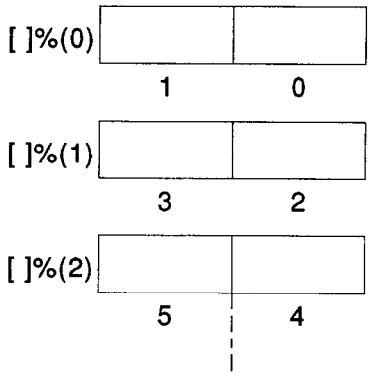
- **<Offset 1>** is used to specify which byte of the variable used as the **<distination storage area>** is to be the head byte where storage of the read-in data will begin.
  <Offset 2> is used to specify which part of the data specified by **<read source>** is to be made the head byte where data read-in will begin.

Specifying an integer variable



An integer variable is composed of two bytes (16 bits). An offset of 0 specifies the lower byte; an offset of 1 specifies the upper byte.

Specifying an integer array



Each element of an integer array is composed of two bytes (16 bits). The offset value is specified as follows.

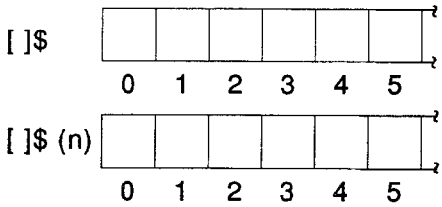
- To specify the lower byte of array
  element number n:

  <offset>= n × 2

- To specify the upper byte of array
  element number n:

  <offset>= n × 2 + 1

Specifying a character string variable or a character string array



Each character within a string variable or array is one byte. To specify the nth character within a variable, the following equation is used:

<offset>= n - 1

- The amount of data to be read in starting at the head position specified with **<offset 2>** is specified in byte units with **<number of bytes>**.
  If an integer variable or array is specified for destination storage area,since each integer element (variable) can store two bytes of data, the storage capacity of n elements is described by the following equation:

  <number of bytes> = n × 2

- If a string variable or array is specified for destination storage area, since each character can store one byte of data, the storage capacity of n characters is described by the following equation:

  <number of bytes> = n

- The position, size and type of data in the buffer memory, common memory, etc. is managed by the user.
  When data is read from or written to buffer memory, common memory, etc. no type conversion is performed.

- When programs are running under multitasking, and it is possible that several programs may try to read or write to the same range of buffer memory, exclusive access management should be performed using the **ZRESERVE** and **ZRELEASE** commands.
  If a program tries to read or write using an area of common memory that is already being read from or written to, an error does not occur. Common memory may be simultaneously accessed.

REMARK

See PUTMEM. Also see sections 4.3 and 8.5.

- Branches to a subroutine and returns to a specified line number.

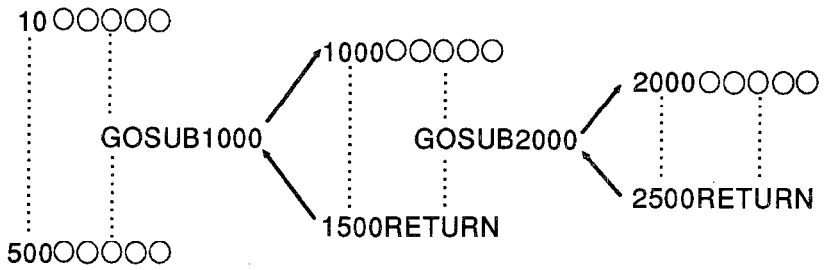
GO to SUBroutine/RETURN

SYNTAX	**GOSUBΔ<line number 1>** **RETURNΔ[<line number 2>]**

	line number 1	Specifies the starting line number or label of the routine to which execution will be branched.
	line number 2	Specifies the line number or label to which execution will be returned.
EXAMPLE	**GOSUB 100**	Execution is branched to the subroutine at line 100.

DESCRIPTION

- The **GOSUB** command branches to a subroutine, and the **RETURN** statement returns execution to the command following the **GOSUB** command or to the specified line number.
- A label can be used in place of an actual line number.
- The same subroutine can be called any number of times within the program.
- Subroutines can be nested as long as memory is sufficient. If available memory is exceeded, an **"Out of memory"** error occurs.



- The **RETURN** command returns control to the command following the **GOSUB** command which caused the branch to the subroutine. If a line number is specified with the **RETURN** command, control is returned to the specified line number.
- Several **RETURN** commands may be included in one subroutine, but they must correctly correspond with the **GOSUB** command.
- Subroutines must always be executed with a **GOSUB** command. If a subroutine is executed independently, when the **RETURN** command is executed, a **"RETURN without GOSUB"** error occurs.

PROGRAMMING EXAMPLE

```
10   'Example using a subroutine
20   PRINT "Currently at line 10 in the main routine"
30   GOSUB 100                                  :'Branch to a subroutine
40   PRINT
50   PRINT "Currently at line 50 in the main routine"
60   END                                        :'End execution
100   PRINT
110   PRINT "Now in the subroutine"
120   RETURN                                     :'Return to the main routine

RUN
Currently at line 10 in the main routine

Now in the subroutine

Currently at line 50 in the main routine
OK
```

REMARK

See section 3.9.

- Unconditionally shifts the flow of program execution to the specified line.

GOTO

SYNTAX	**GOTO <line number>**	
	line number	Specifies the jump destination line number. A label may also be specified.
EXAMPLE	**GOTO 100**	Program execution is shifted to line 100.
	GOTO *IN	Program execution is shifted to the line with label "*IN".

DESCRIPTION

- The **GOTO** command unconditionally shifts the flow of program execution to the specified line.
- Execution is shifted to the line specified by **<line number>**.
- A label may also be specified for **<line number>**.
- If the specified **<line number>** does not exist, an **"Undefined line number"** error occurs.

PROGRAMMING EXAMPLE

```
10  'Repeat the calculation until X becomes 4096
20  X=1                                  :'Define variable X as 1
30  PRINT "X=";X                         :'Display
40  X=2*X                                :'Double the value of X
50  IF X=4096 THEN END                   :'If X equals 4096, end
60  GOTO 30                              :'Return to line 30
RUN
X= I
X= 2
X= 4
X= 8
X= 16
X= 32
X= 64
X= 128
X= 256
X= 512
X= 1024
X= 2048
```

REMARK

See FOR to NEXT, IF to THEN, and IF to GOTO. Also see section 3.6.

- Converts a decimal (base 10) number into a hexadecimal number stored as a character string.

HEXadecimal $

SYNTAX	**HEX$ (<numeric expression>)**	
	numeric expression	Must be specified in the range -32768 to 65535. However, values in the range 32768 to 65535 return the same value as those in the range -32768 to -1 (8000H to FFFFH).
EXAMPLE	**PRINT HEX$(200)**	The number 200 is converted into a hexadecimal number (C8), represented as character string and displayed.

DESCRIPTION

- The **HEX$** function converts a decimal (base 10) number into a hexadecimal number represented as a character string.
- The decimal section of value is truncated to convert it to an integer before it is converted to a hexadecimal number.
- A conversion table for decimal (base 10) and hexadecimal (base 16) numbers is included below.

hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

PROGRAMMING EXAMPLE

```
10 'Convert a decimal (base 10) number to a hexadecimal number.
20 A=200                                    :'Define the number
30 PRINT "decimal =";A
40 A$=HEX$(A)                               :'Convert to hexadecimal
50 PRINT "hexadecimal =";A$
60 END

RUN
decimal= 200
hexadecimal= C8
```

REMARK

See VAL, BIN$, OCT$ and section 2.7.2.

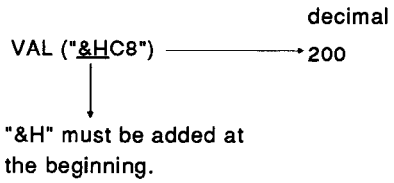
Hexadecimal numbers are indicated by the characters "&H" before the number.

Example

decimal		hexadecimal
200	→	&HC8

The VAL function is used to convert a hexadecimal number represented as a character string into a decimal number.

Example



- Selects a location to branch to based on the results of an expression.

IF~GOTO~ELSE

SYNTAX	**IFΔ<conditional expression>ΔGOTO<line number>[ΔELSE<command>/<line number>]**

conditional expression	A relational operation (<,>,=) expression is specified.
line number	Specifies the line number to which execution is branched if the conditional expression is true.
command/line number	Specifies the command to be executed or the line number to which execution is branched if the conditional expression is false.

EXAMPLE

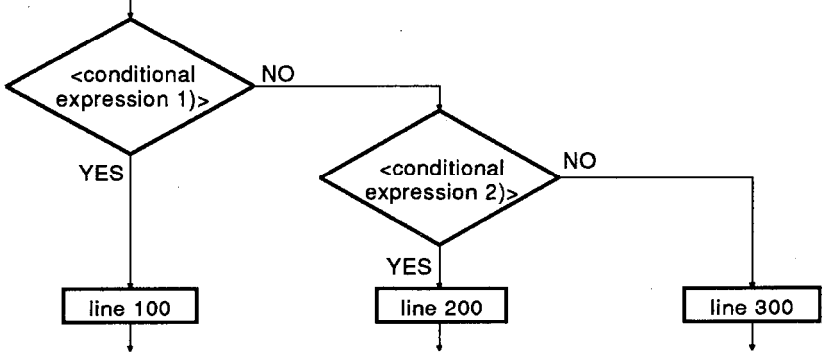
IF X>Y GOTO 100 ELSE END	If X is larger than Y, execution is branched to line 100. If not, program execution is ended.
IF X=0 GOTO 100 ELSE 200	If X is zero, execution is branched to line 100. If x is not zero, execution is branched to line 200.

DESCRIPTION

- The **IF~GOTO~ELSE** command selects a command to execute based on the results of an expression.
- If the **<conditional expression>** is true, the **GOTO** command is executed, and the commands following **ELSE** are ignored.
- If the **<conditional expression>** is false, the **GOTO** is ignored, and the commands following **ELSE** are executed.
  If ELSE is not specified, the command on the next line is executed.
- A label can also be specified for **<line number>**.
- By entering another **IF** command following the **ELSE**, nesting is possible.

Example

IF <conditional expression 1)> GOTO [100] ELSE IF <conditional expression 2)> GOTO [200] ELSE [300]



PROGRAMMING EXAMPLE

```
10 'Branch based on a condition
20 INPUT "X=";X                              :'Input a value
30 IF X>=0 AND X<=10 GOTO 50 ELSE 60         :'If X is between 0 and 10 go to line 50
50 PRINT "In the range 0 to 10!":END         :'       otherwise go to line 60
60 PRINT "Outside the range!":END

RUN
X=? 1
In the range 1 to 10!
OK
RUN
X=? 20
Outside the range!
OK
```

REMARK

See section 3.7.

If the <conditional expression> is an equality between single or double precision real numbers, sometimes correct results cannot be obtained.

For details, see the note in section 3.7.2.

- Selects a command to execute based on the results of an expression.

IF~THEN~ELSE

SYNTAX	**IFΔ<conditional expression>ΔTHEN<command>/<line number> [:<command>/<line number>][ΔELSE<command>/<line number>]**

conditional expression	A relational operation (<,>,=) expression is specified.
command/line number	Specifies the command to be executed or the line number to which execution is branched following the THEN and ELSE.

EXAMPLE

IF X=Y THEN PRINT "EQUAL!" ELSE PRINT "NOT EQUAL!"
If X equals Y, "EQUAL!" is displayed. If not, "NOT EQUAL!" is displayed.

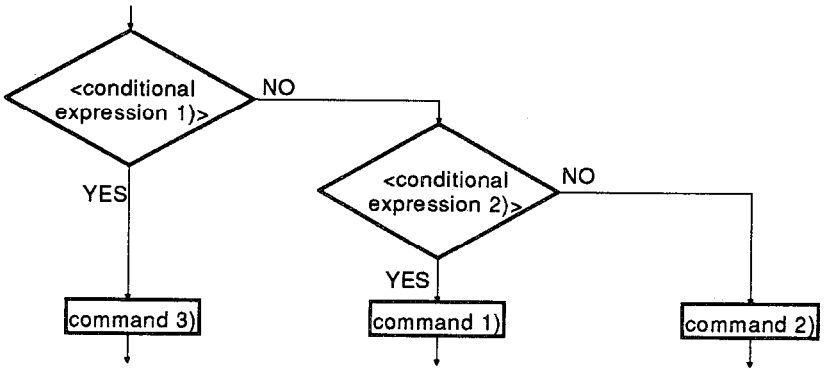
IF X=0 THEN 100 ELSE 200
If X is zero, execution is branched to line 100. If X is not zero, execution is branched to line 200.

DESCRIPTION

- The **IF~THEN~ELSE** command selects a command to execute or a line number to branch to based on the results of an expression.
- If the **<conditional expression>** is true, the commands following **THEN** are executed, and the commands following **ELSE** are ignored.
- If the **<conditional expression>** is false, the commands following **THEN** are ignored,and the commands following **ELSE** are executed.
  If **ELSE** is not specified, the command on the next line is executed.
- A label can also be specified for **<line number>**.
- By entering another **IF** command following **THEN** or **ELSE**, nesting is possible.

Example

IF <conditonal expression 1)> THEN IF <conditional expression 2)> THEN [COMMAND 1)] ELSE [COMMAND 2)] ELSE [COMMAND 3)]



PROGRAMMING EXAMPLE

```
0  'Select a command to execute based on a condition
20 INPUT "X=";X                                  :'Input X
30 INPUT "Y=";Y                                  :'Input Y
40 IF X=Y THEN PRINT "SAME"
   ELSE PRINT "DIFFERENT"                        :'If X = Y, display "SAME";
50 END                                           :'       if X <> Y, display "DIFFERENT"

RUN
X=? 5
Y=? 5
SAME
OK
RUN
X=? 3
Y=? 9
DIFFERENT
OK
```

REMARK

If the <conditional expression> is an equality between single or double precision real numbers, sometimes correct results cannot be obtained.

For details, see the note in section 3.7.2.

- If there is any keyboard input, the function returns the character of the key pressed; if there is no keyboard input, the function returns the empty string.

INput KEY $

SYNTAX	**INKEY$**	
EXAMPLE	**A$ = INKEY$**	If there is any keyboard input, the character of the key pressed is stored in A$; if there is no keyboard input, the empty string is stored in A$.

DESCRIPTION

- The **INKEY$** function returns the character of the key pressed if there is any input from the keyboard specified with the **ZIDV** command, ; if there is no keyboard input, the function returns the empty string. The character is not printed on the screen.
- All keys except those shown below can be input from the keyboard set as the console.
  [Ctrl]+[C], [Ctrl]+[S], [Break] (these keys perform their normal functions)
  Keys with corresponding character codes 00_H, 80_H or FD_H to FF_H
- All keys except the key corresponding to code 00_H, and the keys assigned the codes DC1/DC3 with the **ZCNTL** command can be input from a non-console keyboard (i.e. a console attached to the communication port).
- Commands such as the carriage return character which cannot be input using the **INPUT** and **LINE INPUT** commands can be input using the **INKEY$** function.

POINT

When input is being received from a non-console keyboard (i.e. a console attached to the communication port), if the 03_H code (BREAK code) is input, the BASIC program handles it as data. Therefore, the BASIC program cannot be stopped from the keyboard. Exercise caution when using non-console keyboards.

PROGRAMMING EXAMPLE

```
10 'End the program if a key is pressed.
20 PRINT "*";
30 A$=INKEY$                                  :'Scan the keyboard
40 IF A$="" GOTO 20                           :'If no key was pressed, return to line 20
50 END

RUN
****************
OK
```

REMARK

See INPUT$, CONT and INPUT. Also see section 3.11.

- Performs input of data from the keyboard.

INPUT

SYNTAX	**INPUTΔ["<display string>";] <variable name> [,<variable name>, ···]**

	display string	The specified a character string constant is displayed when input is requested.
	variable name	Specifies the variable in which to store the data that is input.
EXAMPLE	**INPUT "Target number="; M**	The string "Target number =" is displayed on the console screen and the program waits for key input. When data is input, it is stored in variable M.

DESCRIPTION

- When the **INPUT** command is executed, program execution is temporarily stopped, and a question mark is displayed on the screen specified by the **ZODV** command. Once the question mark is displayed, input of data can be performed from the keyboard specified by the **ZIDV** command.
- If **<display string >** is specified, it is displayed before the question mark on the screen specified by the **ZODV** command.
- The data that is input is stored in the variables specified for **<variable name>.**
- The number of data items that are input must be the same as the number of **<variable names>** specified.
  Numeric variables or character string variables may be specified for **<variable name>.**
  Each data item that is input must be of the same data type as the corresponding **<variable name>.**
  (It is not necessary to enclose character string input in quotes when using the INPUT command.)
  If the number of data items input or the types of the data items input do not match the variables used in the INPUT command, the message **"?Redo from start"** (start again) is displayed, and data input is performed again.
  Also, if the number of data items input is larger than the number of variables specified in the INPUT command, the message **"Extra ignored"** is displayed, and execution continues to the next command.
- Data items must be separated by a comma.
- The **INPUT** command waits for input until the [Enter] key is pressed.
- The comma and quote characters cannot be input with the **INPUT** command. The **LINE INPUT** command is used to input these characters.

- If the **ZIDV** command is used to redirect input from a device other than the terminal specified as the console by the AD51H system dip switch, when the following keys are pressed, their key codes are stored as is to the variable specified with **<variable name>**.
  [Ctrl]+[C], [Ctrl]+[S], [Break]
  The keys assigned the codes DC1/DC3 with the **ZCNTL** command.
  Therefore, program execution cannot be stopped by pressing [Ctrl]+[C] or [Break].

  Exercise caution when using non-console keyboards.

PROGRAMMING EXAMPLE

```
10 'Perform data input
20 INPUT "X="X                    :'Input
30 PRINT "X^2=";X^2
40 END

RUN
X=? 2
X^2=  4
OK
```

REMARK

See INPUT$, LINE INPUT. Also see section 3.11.

INPUT $

- Returns as its value a character string of a specified length which contains data input from the keyboard.
- Returns as its value a character string of a specified length which contains data input from a sequential file.
- Returns as its value a character string of a specified length which contains data input from the AD51H communication port.

SYNTAX

INPUT$ (<numeric expression>)
INPUT$ (<numeric expression>, # <file number>)
INPUT$ (<numeric expression>, %<port number>[, <wait time>])

numeric expression	Specifies the length of the character string data to read.
file number	Specifies the file number assigned to a sequential file with the OPEN command.
port number	Specifies the port number assigned to a port with the ZOPEN command.
wait time	Specifies the amount of time in seconds to wait for input from the communication port.

EXAMPLE

A$=INPUT$(3)	Three characters are read from the console keyboard and stored in A$.
B$=INPUT$(10,#1)	Ten characters are read from the file opened as file #1 and stored in B$.
C$=INPUT$(7,%1,20)	Seven characters are read from AD51H channel 1 (RS-232C1) and stored in C$. If after 20 seconds, seven characters have not been read in, a "Time out error" occurs.

DESCRIPTION

- The **INPUT$** function reads a certain number of characters specified with **<numeric expression>** from the keyboard, sequential file or communication port and returns the read-in data as its value.
- The number of characters of data to read is specified with **<numeric expression.>**
  The value specified for **<number expression>** must be in the range 1 to 255.

READING FROM THE KEYBOARD

- If nothing is specified following **<numeric expression>**, input is received from the keyboard specified by the **ZIDV** command.

- When input is received from the keyboard designated as the console, all keys and key codes can be read except the following:
  [Ctrl]+[C],[Ctrl]+[S],[Break] (these keys perform their normal functions)
  00_H, 80_H or FD_H to FF_H key codes

- When reading from a keyboard other than the console (i.e. a terminal attached to the communication port), all keys except the key corresponding to code 00H, and the keys assigned the codes DC1/DC3 with the **ZCNTL** command can be handled.

POINT

When input is being received from a non-console keyboard (i.e. a console attached to the communication port), if the 03_H code (BREAK code) is input,the BASIC program handles it as data. Therefore, the BASIC program cannot be stopped from the keyboard. Exercise caution when using non-console keyboards.

- When input is read from the keyboard, the program continues waiting until the number of characters specified by **<numeric expression>** have been entered.
  However, if previously entered characters remain in the buffer, the INPUT$ function reads from the buffer first.
  The characters that are input are not displayed on the screen.

READING FROM A SEQUENTIAL FILE

- If a **<file number>** is specified following the **<number expression>**, the number of characters specified by <number expression> are read from a sequential file.

- **<File number>** represents the file number assigned to the file to be read from when the file was opened with the **OPEN** command.

- When reading from a file with the INPUT$ function, even the delimiter symbol characters are read as data.

- When reading from a file, if there is no more data to read, an **"Input past end"** error occurs.

READING FROM AN AD51H COMMUNICATION PORT

- If a **<port number>** is specified following the **<numeric expression>**, data is read from one of the AD51H communication ports- CH1 (RS-232C), CH2 (RS-232C) or CH3 (RS-422).

- The port number should be specified according to the following chart:

  CH1 (RS-232C) 1

  CH2 (RS-232C) 2

  CH3 (RS-422) 3

- All characters and key codes including 00H can be read in through the communication port as data.
  Therefore, even if the 03H code (BREAK code) is entered, the BASIC program handles it as data, so it is impossible to stop BASIC program execution externally. Exercise caution when reading from the communication port.

- The amount of time to wait for data input can be specified with <wait time.>
  Values from 0 to 255 may be used (values represent seconds).
  If the number of characters of data specified with **<numeirc expression>** have not been input in the time specified, a **"Time out error"** occurs.
  If **<number expression>** is set to 0, or if **<number expression>** is omitted, the program will continue waiting indefinitely until the number of characters of data specified by **<number expression>** have been input.

- When reading from the port, if any previously entered data exists in the buffer,the buffer data is read first.

PROGRAMMING EXAMPLE

```
10 'Waiting for input from the keyboard
20 PRINT "Yes or No ?";
30 A$=INPUT$(1)                                :'Input one character from the keyboard
40 IF A$="Y" OR A$="y" THEN PRINT:PRINT"OK!!":GOTO 60
50 PRINT:PRINT "???"
60 END

RUN
Yes or No ?
OK !!
OK
RUN
Yes or No ?
???
OK
```

REMARK

See INPUT and LINE INPUT. Also see section 3.11.

- Reads data from a sequential file.

INPUT #

SYNTAX	**INPUT Δ # <file number>, <variable name> [, <variable name>,·····]**

	file number	Specifies the file number assigned to the file when it was opened with the OPEN command.
	variable name	Specifies the variable in which to store the data that is read in.
EXAMPLE	**INPUT #2, A**	Numeric data is read from the file opened as file number 2 and stored in A.

DESCRIPTION

- The **INPUT #** command reads data from a sequential data file and stores it in a variable.
- **<File number>** represents the file number assigned to the sequential file that is to be used for input when the file was opened with the **OPEN** command.
- **<Variable name>** indicates the variable(s) where the data items that are read from the file will be stored. The type of data read from the file must match the type of variable used.
- If the data item is a numeric value, the first character that is not whitespace, enter, or linefeed is considered the first character of the numeric value. The end of the numeric value is signified by whitespace, enter, line feed or a comma.
- If the data item is a character string, the first character that is not whitespace, enter, or linefeed is considered the first character of the string.
  However, if the first character of the string is a quote ("), the character string is composed of all characters between the first quote character and the next quote character.
  If the string is not enclosed in quotes, the end of the string is signified by the enter code, or by a comma.
  Up to 255 characters can be read as one data item.
  If there is no delimiter character (enter, comma) within the first 255 characters, the data item is automatically read as the first 255 characters.
- If the end of the file is reached while reading a data item, the data item is assumed to end at the end of the file.

REMARK

See INPUT$, LINE INPUT#, LOF and OPEN. Also see chapter 6.

- Searches within a string for a specified substring, and if it exists, returns the starting position of the string.

IN STRing

SYNTAX	**INSTR ([numeric expression,] <character string 1>, <character string 2>)**

numeric expression	Specifies the position in the string from which to begin the search.
character string 1	Specifies the string to be searched.
character string 2	Specifies the string to be searched for.

EXAMPLE

A=INSTR (2, A$, "ABC")

Starting from the second character of A$, the string "ABC" is searched for. If it is found, the position within A$ where it is located is returned as the function value; if it is not found, zero is returned as the function value.

range to be searched

	1	2	3	4	A	B	C	D	E	F		
A$	1	2	3	4	A	B	C	D	E	F	→	A 5

	1	2	3	4	5	6	7	8	9	10		
A$	A	B	1	2	3	4	C	D	E	F	→	A 0

DESCRIPTION

- The **INSTR** function searches for **<character string 2>** within **<character string 1>**.
  If it is found, the function returns its starting position within **<character string 1>**; if it is not found, the function returns zero.
- **<Numeric expression>** specifies the position within **<character string 1>** to start the search. If expression is omitted, the search begins with the first character of **<character string 1>**; if the value specified for expression is larger than the number of characters in **<character string 1>**, the function returns the value zero.
- If **<character string 1>** is the empty string, the function returns the value zero.
- If **<character string 2>** is the empty string, the function returns the value that is the same as **<numeric expression>**.

PROGRAMMING EXAMPLE

```
10 'Search for the specified character
20 A$="MITSUBISHI AD51H SEQUENCER"
30 A=INSTR (2, A$, "SEQUENCER")              :'Search for "SEQUENCER"
40 PRINT "A$=";A$                            :'Display the character
50 PRINT "SEQUENCER begins at the ";A;"the character.":'Display the location where
                                              "SEQUENCER" was found
60 END

RUN
A$=MITSUBISHI AD51H SEQUENCER
SEQUENCER begins at the 10th character.
OK
```

- Returns the integer value of a numeric expression.

INTeger

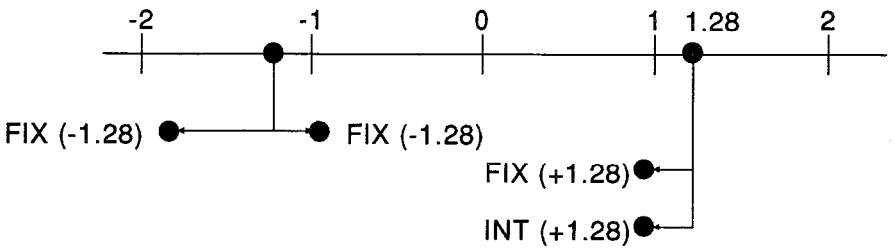
SYNTAX	**INT (<numeric expression>)**	
	numeric expression	Specifies the value to be the object of the operation.
EXAMPLE	**A = INT (B)**	The value of B is stored to A as an integer.

DESCRIPTION

- The **INT** function returns the largest integer that does not exceed the value of **<numeric expression>**.
- The **INT** function removes the decimal portion of <numeric expression> by returning the largest integer that does not exceed **<numeric expression>**. Therefore, when the value of expression is negative, the results of the INT function differ from the results of the **FIX** function, which simply truncates <numeric expression> at the decimal point.

Example

FIX (1.28)	1	INT (1.28)	1
FIX (-1.28)	-1	INT (-1.28)	-2



- Type conversion does not occur as with the **CINT** function.

PROGRAMMING EXAMPLE

```
10 'The difference between the calculation methods of the INT, FIX and CINT functions.
20 DIM A(3)                                  :'Define an array
30 A(0)=12.34                                :'Store numeric values
40 A(1)=12.56
50 A(2)=-12.34
60 A(3)=-12.56
70 FOR i=0 TO 3                              :'Loop from I = 0 to I = 3
80  PRINT "A(";I;")=";A(I)                   :'Display the original numeric value
90  PRINT "INT=";INT(A(I))                   :'The results of the INT function
100  PRINT "FIX=";FIX(A(I))                  :'The results of the FIX function
110  PRINT "CINT=";CINT(A(I))                :'The results of the CINT function
120  PRINT
130 NEXT I
140 END

RUN
A(0)=12.34
INT = 12
FIX = 12
CINT= 12

A(1)=12.56
INT = 12
FIX = 12
CINT= 12

A(2)=-12.34
INT =-13
FIX =-12
CINT=-13

A(3)=-12.56
INT =-13
FIX =-12
CINT=-13

OK
```

REMARK

See CINT and FIX.

- Assigns a character string to the console's function keys.

KEY

SYNTAX	**KEYΔ<key number>, <character string>**	
	key number	Specifies the key to which the character string is to be defined.
	character string	Specifies the character string to be assigned.
EXAMPLE	**KEY 1, "MITSUBISHI"**	The string "MITSUBISHI" is assigned to the function key corresponding to console function key number 1 ([F1]).

DESCRIPTION

- The **KEY** command assigns a character string or control character to a console function key.
- The value specified for **<key number>** corresponds with the numbers of the function keys on the keyboard.
- **<Character string>** specifies the character string to be assigned to the specified function key. It can be up to 15 characters long.
  If more than 15 characters are specified, only the first 15 characters are assigned to the key.
  Control characters must be assigned using the **CHR$** function.
- Once a key has been defined, when the key is pressed, operation is exactly as if the specified character string had been typed on the keyboard.
- The **KEYLIST** command can be used to display the assignments for all function keys on the screen.
- Each function key is predefined with the following character strings:

[F1] ..... **load"**	[F6] ..... **save"**
[F2] ..... **auto**	[F7] ..... **key**
[F3] ..... **goto**	[F8] ..... **print**
[F4] ..... **list**	[F9] ..... **list. +0D$_H$+1E$_H$+1E$_H$**
[F5] ..... **run+0D$_H$**	[F10] .... **cont+0D$_H$**

REMARK

See KEYLIST.

- Displays the character strings that are assigned to the console function keys.

KEY LIST

SYNTAX	**KEYLIST**

EXAMPLE	**KEYLIST**	The character strings that are assigned to the console function keys are displayed.

DESCRIPTION

- The **KEYLIST** command displays the character strings that are assigned to the console function keys **F1 to F10**.
- Each function key is predefined with the following character strings:

[F1]	**load"**	[F6]	**save"**
[F2]	**auto**	[F7]	**key**
[F3]	**goto**	[F8]	**print**
[F4]	**list**	[F9]	**list. +0D$_H$+1E$_H$+1E$_H$**
[F5]	**run+0D$_H$**	[F10]	**cont+0D$_H$**

- The character strings that are assigned to each function key can be defined with the **KEY** command.

REMARK

See KEY.

- Deletes a file from a memory card, floppy disk or hard disk.
- Deletes a system name from a memory card, floppy disk or hard disk.

KILL

SYNTAX	**KILL " [<drive number>:] [<system name>\] [<file name>] "**

	drive number:	Specifies the memory card or disk where the file to be deleted is stored.
	system name\	Specifies the system name where the file to be deleted is written.
	file name	Specifies the file name of the file to be deleted.
EXAMPLE	**KILL "1:TEST \No0.BAS"**	The file named "NO1.BAS" stored under system name "TEST" on the memory card inserted in memory card slot 2 is deleted.
	KILL "0:BASIC \"	The system name "BASIC" on the memory card inserted in memory card slot 1 is deleted.

DESCRIPTION

- The **KILL** command deletes a file from a memory card inserted in the memory card interface, or from a floppy disk or hard disk.
- The memory card or disk where the file to be deleted is stored is specified with **<drive number>**. The following table shows the correspondence between drive numbers and actual devices.

EDIT MODE OR EXECUTE MODE

Memory Card 1 0

Memory Card 2 1

PC/AT FDD 2

PC/AT HDD[C] 3

OFFLINE PROGRAMMING

PC/AT FDD 1

PC/AT HDD[C] 3

- **<System name>** specifies the system name under which the file to be deleted is written. If **<system name>** is omitted, files with no system name can be specified.
- **<File name>** specifies the name and file extension of the file to be deleted. The file extension cannot be omitted when using the **KILL** command.

- The **KILL** command cannot be executed on an open file. If it is executed on an open file, a **"File already open"** error occurs.

REMARK

See section 3.4.3.

- Reads a substring of a specified length from a character string starting with the leftmost character of the string.

LEFT $

SYNTAX	**LEFT$ (<character string expression>, <numeric expression>)**

	character string expression	Specifies the character string from which the substring expression is to be read.
	numeric expression	Specifies the length of the substring to be read from the main string.
EXAMPLE	**A$=LEFT$ (B$, 5)**	The first five characters of character string B$ are read and stored in A$.

B$ | M | I | T | S | U | B | I | S | H | I |

5 characters from the left

↓

A$ | M | I | T | S | U |

DESCRIPTION

- The **LEFT$** function reads a substring of a specified length from a character string starting with the leftmost character of the string and returns the substring as its value.
- If the value of **<numeric expression>** is larger than the length of **<character string expression>**, the entire character string is returned.
- The acceptable range of values for **<numeric expression>** is 0 to 255. If a value greater than 255 is specified, an **"Illegal function call"** error occurs; if zero is specified, the empty string ("") is returned.
- If the substring is to be stored in the same variable as the string it is read from, following the operation of the LEFT$ function, the variable contents are overwritten.

Example

A$=LEFT$ (A$, 5)

A$ | A | B | C | D | E | F | G | H | I | J |

5 characters from the left

↓

A$ | A | B | C | D | E |

PROGRAMMING EXAMPLE

```
10 'Extract the first five characters starting from the left of a character string.
20 A$="AD51H-BASIC"                          :'Define the character string
30 B$=LEFT$(A$, 5)                           :'Extract the leftmost five characters
50 PRINT "The leftmost 5 characters ---> ";B$ :'Display the extracted substring
60 END

RUN
The leftmost five characters --> AD51H
OK
```

REMARK

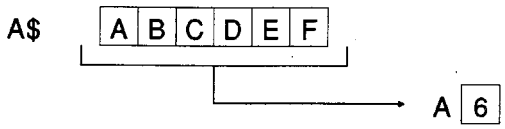
See RIGHT$, MID$(version 2) and KMID$.

- Returns the number of characters in a character string.

LENgth

SYNTAX	**LEN (<Character string expression>)**

	character string expression	Specifies the character string whose length is to be calculated.
EXAMPLE	**A = LEN (A$)**	The number of characters in character string variable A$ is stored in A.

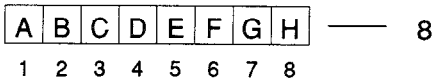


DESCRIPTION

- The **LEN** function counts the number of characters in the character string specified for character string expression.
- Characters with character codes in the range 00_H to $1F_H$ (control characters) and whitespace characters are counted.
- **LEN** function counts a double size character as two single size characters.

Example

LEN function



PROGRAMMING EXAMPLE

```
10 'Finds the length of a character string
20 A$="MITSUBISHI AD51H-BASIC"              :'Define a character string
40 PRINT "Length -->";LEN(A$)
50 END

RUN
Length -->15
OK
```

- Assigns the value of an expression to a variable.

LET

SYNTAX	**LETΔ<variable name> = <expression>**

	variable name	A numeric variable, character string variable or array can be specified.
	expression	A numeric expression, character string, array or function can be specified.
EXAMPLE	**LET A=1**	The value 1 is assigned to the numeric variable A.
	LET A$="MITSUBISHI"	The character string "MITSUBISHI" is assigned to the character string variable A$.

DESCRIPTION

- The **LET** command assigns the value of an expression to a variable.
- **The LET command itself can be omitted.**

  Example

  LET A=10 ..... A=10
  LET A$="NAGOYA" ..... A$="NAGOYA"

- If **<expression>** is a numeric expression, **<variable name>** must be a numeric variable; if **<expression>** is a character string, **<variable name>** must be a character string variable. If **<expression>** and **<variable name>** are different types, a **"Type mismatch"** error occurs.
- If **<expression>** and **<variable name>** are of different numeric formats, **<expression>** is converted to the same format as **<variable name>** before the value is assigned.

  Example

integer variable	**A%=55.88** **PRINT A% 55**
single precision variable	**B=.8571428571428571** **PRINT B ..... .857143**
double precision variable	**C#=2.04** **PRINT C# 2.039999961853027**

PROGRAMMING EXAMPLE

```
10  'Assign the value 1 to variable A, and assign the expression A + 1 to variable B
20  A=1                                        :'Store 1 in A
30  B=A+1                                      :'Store A + 1 in B
40  PRINT "A=";A,"B=";B                        :'Display A and B
50  END

RUN
A=1             B=2
OK
```

REMARK

See section 3.5.1.

- Prints the names of the files on a memory card, floppy disk or hard disk to the printer.

LFILES

SYNTAX	**LFILESΔ[" [<drive number>:] [<system name>\] "] [,S]**	
	drive number	Specifies the memory card or disk whose file names are to be printed.
	system name	Specifies the system name where the files to be printed are stored.
EXAMPLE	**LFILES "0:",S**	The name, size, and date and time of creation of all files with no system name on the memory card inserted in memory card slot 1 are printed on the printer.
	LFILES "2:TEST\"	The names of all files stored in system name "TEST" on the PC/AT floppy drive are printed on the printer.

DESCRIPTION

- The **LFILES** command prints the names of all files in the specified **<system name >** on the memory card, floppy disk, hard disk, etc. specified by **<drive number>** to the printer specified with the **ZLDV** command.
- The following table shows the **<drive number>** specifications and their corresponding storage devices.

  ON-LINE PROGRAMMING

  AD51H Memory Card [1] 0

  AD51H Memory Card [2] 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFFLINE PROGRAMMING

  PC/AT FDD 1

  PC/AT HDD[C] 3
- If the **[,S]** option is specified, the file name, size, and date and time of creation are printed.
  If the **[,S]** option is omitted, only the file names are printed.
- If **<system name>** is omitted, all files with no system name are printed.

POINT

When output is being sent to the printer, all keyboard input including the [Break], [Ctrl]+[C] keys is disabled. Thus, to abort printer output, it is necessary to turn off the printer power or to set the printer to OFFLINE condition. After about five seconds, the printer output operation is aborted.

REMARK

See FILES. Also see section 3.4.1.

- Reads an entire line of keyboard input (up to 255 characters) and stores it all, without any separation, to a character string variable.

LNE INPUT

SYNTAX	**LINE INPUΔ["<display string>";]string variable**

	display string	Specifies a string to be displayed before the input operation.
	string variable	Specifies the string variable to which the input will be stored.
EXAMPLE	**LINE INPUT "DATA-";A$**	Data is input from the keyboard until the ENTER key is pressed, and then the data is stored to the variable A$.

DESCRIPTION

- The **LINE INPUT** command temporarily suspends program execution and accepts input from the keyboard specified with the **ZIDV** command.
- The **LINE INPUT** command, unlike the **INPUT** command, does not display a question mark.
- When the **LINE INPUT** command is used, all characters input up until the [Enter] key is pressed are stored in <string variable>.
- Keyboards from which key input can be accepted are as follows:
  CONSOLE KEYBOARD
  KEYBOARD OF A TERMINAL CONNECTED TO THE COMMUNICATION PORT
- If <display string> is specified, the specified character string is displayed on the screen designated by the **ZODV** command before the input operation is performed.
- If no key input is entered, and the RETURN key is pressed, the empty string is stored to **<string variable>.**
- To input data from a terminal attached to a communication port, the port must first be opened with the **ZOPEN** command, and then the input device must be designated with the **ZIDV** command.
- If the **ZODV** command is used to receive input from a source other than the console designated by the AD51H system dip switch, the following keys are stored to <string variable> as ordinary data:
  [Ctrl]+[C], [Ctrl]+[S], [Ctrl]+[Q], [Break]
  Therefore, program execution cannot be stopped with the [Ctrl]+[C] or [Break].
  Care should be taken when using such keyboards.

PROGRAMMING EXAMPLE

```
10  'Input a character string until the enter key is pressed.
20  LINE INPUT "Address : ";A$
30  LINE INPUT "Name    : ";B$                    :'Input a character string
40  PRINT
50  PRINT A$;" : ";B$                             :'Display the character string that was input
60  END

RUN
Address : NAGOYA
Name    : MITUBISHI
NAGOYA : MITUBISHI
OK
```

REMARK

See CLOSE, INPUT and INPUT$.

- Reads an entire line (255 characters) without separation from a sequential file and stores it in a character string variable.

LINE INPUT #

SYNTAX	**LINE INPUTΔ#<file number>,<string variable>**

	file number	Specifies the file number of the file from which data will be read.
	string variable	Specifies the variable into which the data that is read will be stored.
EXAMPLE	**LINE INPUT #1,A$**	One line is read from the sequential file with file number 1 and stored in variable A$.

sequential file #1

ABCDEFG ..... ⇨ A$="ABCD "

255 characters

DESCRIPTION

- The **LINE INPUT #** command reads an entire line without separation from the sequential file specified by **<file number>** and stores it in <string variable>.
- To use this command, it is first necessary to open the sequential file with the **OPEN** command.
- The **LINE INPUT #** command reads all characters up until the enter character (&H0D) without separation. Any characters including commas, colons, semicolons and quotes may be included.

"ABCD","EF:GH","IJ;KL" ⇨ A$

22 characters

- Up to 255 characters may be read with one LINE INPUT # command. If no ENTER key is encountered within 255 characters, the string is considered to consist of all the characters up to the 255TH.

ABCDEFGH [Enter]	The 8 characters up to the [Enter] character are read in.
ABCDEF .....; "XY"	All 255 characters are read in.

REMAKE

See INPUT# and INPUT$. Also see chapter 6.

- Displays all or part of the program stored in the program area.

LIST

SYNTAX	**LISTΔ[<line number 1> [- [<line number 2>]]]**

	line number 1	Specifies the first line of the program to be displayed.
	line number 2	Specifies the last line of the program to be displayed.
EXAMPLE	LIST	All lines are displayed.
	LIST 500	Only line 500 is displayed.
	LIST 150-	All lines from line 150 to the end of the program are displayed.
	LIST -1000	All lines from the beginning of the program to line 1000 are displayed.
	LIST 150-1000	All lines from line 150 to line 1000 are displayed.
	LIST.	The line where execution was stopped due to a STOP command, an error, etc. is displayed.

DESCRIPTION

- The **LIST** command displays the program stored in the program area.
- If only **<line number 1>** is specified, only the specified line is displayed.
- If **<line number 1>** is specified followed by a hyphen, all lines from the specified line to the end of the program are displayed.
- If **<line number 2>** is specified preceded by a hyphen, all lines from the beginning of the program to the specified line are displayed.
- If both **<line number 1>** and **<line number 2>** are specified, all lines from **<line number 1>** to **<line number 2>** are displayed.
- If both line numbers are omitted, the entire program is displayed.
- If the line number pointer (.) is specified instead of a line number, the line where program execution was stopped due to a **STOP** command, error, etc. is displayed.
- The [Ctrl]+[S] key can be pressed to temporarily stop execution of the **LIST** command.
  Pressing any key restarts execution. The [Ctrl]+[C] or [Break] keys can be pressed to permanently cancel execution of the **LIST** command.

- If the **LIST** command is executed in program mode, after the program is displayed, BASIC enters the "waiting for command input" condition.

REMARK

See LLIST. Also see section 3.2.3.

- Prints all or part of the program stored in the program area on the printer.

LLIST

SYNTAX	**LLISTΔ[<line number 1> [- [<line number 2>]]]**

	line number 1	Specifies the first line of the program to be printed on the printer.
	line number 2	Specifies the last line of the program to be printed on the printer.
EXAMPLE	LLIST	All lines are printed.
	LLIST 100	Only line 100 is printed.
	LLIST 200-	All lines from line 200 to the end of the program are printed.
	LLIST -150	All lines from the beginning of the program to line 150 are printed.
	LLIST 300-1000	All lines from line 300 to line 1000 are printed.

DESCRIPTION

- The **LLIST** command prints all or part of the program stored in the program area on the printer.
- After execution of the **LLIST** command is completed BASIC enters the "waiting for command input" condition.

POINT

When output is being sent to the printer, all keyboard input including the [Break] and [Ctrl]+[C]keys is disabled. Thus, to abort printer output, it is necessary to turn off the printer power or to set the printer to OFFLINE condition. After about five seconds, the printer output operation is aborted.

- The printer to receive output is designated with the **ZLDV** command.

REMARK

See ZLDV and LIST. Also see section 3.2.3.

- Reads in a program from a memory card, floppy disk drive or hard disk drive.

LOAD

SYNTAX	**LOADΔ"[<drive number>:] [<system name>\] <file name>"**

	drive number	Specifies the memory card, floppy drive or hard drive where the program to be loaded is stored.
	system name	Specifies the system name under which the program to be loaded is stored.
	file name	Specifies the name of the program file to be loaded.
EXAMPLE	**LOAD "0:TEST\No1.BAS"**	The program "NO1.BAS", which is stored under system name "TEST" on the memory card in memory card slot 1 is loaded.

DESCRIPTION

- The **LOAD** command loads a program that is stored on a memory card, floppy disk or hard disk into AD51H memory.
- The **<drive number>** specifications correspond to the physical devices as shown below:

  ON-LINE PROGRAMMING

  AD51H Memory Card 1 0

  AD51H Memory Card 2 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFFLINE PROGRAMMING

  PC/AT FDD 1

  PC/AT HDD[C] 3
- **<System name>** specifies which system name the program is stored under. If the specified system name does not exist, a **File not found** error occurs.
- **<File name>** specifies the name and file extension of the file in which the program is stored. A file with no extension can be specified by the file name.

REMARK

See CHAIN, MERGE, RUN and SAVE. Also see section 3.3.

- Returns the current logical position within a file.

LOCation

SYNTAX	**LOC (<file number>)**	
	file number	Specifies the number of the file whose current position is to be returned.
EXAMPLE	**PRINT LOC(1)**	If file number 1 is a sequential file, the current position within the file is displayed in 256-byte units offset from the head of the file. If it is a random file, the number of the last record accessed is displayed.

DESCRIPTION

- The **LOC** function returns the current logical position within a file.
- The value returned by the **LOC** function depends on the type of file that is specified by **<file number>.**

SEQUENTIAL FILE
The function returns as its value the number of records read from or written to the file since it was opened. (one record = 256 bytes).

RANDOM FILE
The function returns as its value the number of the last record accessed with the **GET** or **PUT** commands.

PROGRAMMING EXAMPLE

```
10  'Detects the record number of a random file to which data has been written.
20  OPEN "LOC.DAT" AS #1                        :'Open the file
30  FIELD #1,8 AS CS$,10 AS AR$,4 AS CL$,10 AS NA$:'Assign variable ranges to the random
                                                 file buffer
40  INPUT "CALLSIGN";C$:LSET CS$=C$             :'Input data
50  INPUT "AREA    ";A$ LSET AR$=A$
60  INPUT "CLASS   ";B$:LSET CL$=B$
70  INPUT "NAME    ";N$:LSET NA$=N$
80  PUT #1                                      :'Write to the file
90  PRINT "Record #";LOC(1);"has been written." :'Display the record number written to
                                                 with the PUT command
100  INPUT "Continue writing? (Y/N)";Y$         :'Input Y or N
110  IF Y$="Y" OR Y$="Y" THEN 40                :'If Y was input, continue recording
120  CLOSE #1                                   :'Close the file
```

REMARK

See EOF and LOF.

- Specifies a display position on the console screen.

LOCATE

SYNTAX	**LOCATEΔ<horizontal position>,<vertical position>[,<cursor switch>]**	
	horizontal position	Specifies the horizontal (X-axis) screen position.
	vertical position	Specifies the vertical (Y-axis) screen position.
	cursor switch	Specifies whether or not to display the cursor.
	LOCATE 5,1	The display position at column number 5, line number 1 is specified.
EXAMPLE	**LOCATE 10,20,0**	The display position at column number 10, line number 20 is specified, and the cursor is not displayed.

DESCRIPTION

- The **LOCATE** command moves the cursor to a specified position on the display screen of the console designated by the **ZODV** command.
- **<Vertical position>** and **<horizontal position>** specify the location to which the cursor is to be moved.
- Values in the range 0 to 31 may be specified for **<vertical position>**. However, depending on the display, the number of lines that can be displayed varies.
- Values in the range 0 to 79 may be specified for **<horizontal position>**.

When the **LOCATE** command is executed, the next command which sends output to the screen or reads keyboard input will be performed at the position designated with the **LOCATE** command.

- If the values specified for **<vertical position>** and **<horizontal position>** exceed the range of values that can be displayed by the screen, an error does not occur.
  However, the next screen display will occur at an undefined location.
- If the value 1 is specified for **<cursor switch>**, the cursor is displayed, if 0 is specified, the cursor is not displayed. To display the cursor again, execute the **LOCATE** command.

PROGRAMMING EXAMPLE

```
10  'Display a character string at a specified screen location.
11  CLS                                    :'Clear the screen
12  Y=1
20  A$="AD51-BASIC"                        :'Define the character string
30  FOR X=1 TO 10
40   Y=Y+1
50   LOCATE X,Y                            :'Set the display position
60   PRONT A$                              :'Display the string
80  NEXT X
90  END
```

```
 AD51H-BASIC
  AD51H-BASIC
   AD51H-BASIC
    AD51H-BASIC
     AD51H-BASIC
      AD51H-BASIC
       AD51H-BASIC
        AD51H-BASIC
         AD51H-BASIC
          AD51H-BASIC
OK
```

REMARK

See TAB. Also see section 3.10.2.

- Returns the length of a file in record units (number of sectors).

Length Of File

SYNTAX	**LOF (<file number>)**	
	file number	Specifies the file number established with the OPEN command.
EXAMPLE	**PRINT LOF(1)**	The number of records (sectors) in file number 1 is displayed.

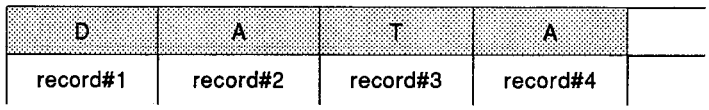
DESCRIPTION

- The **LOF** function returns the length of a file in record units (number of sectors).
- A record is the smallest unit that can be managed on a floppy disk, hard disk or memory card. 256 bytes of data may be stored in one record.
- The value returned by the LOF function depends on the type of file specified by **<file number>**.

SEQUENTIAL FILE

Data is written in byte units, but the file size is returned as a record number (number of sectors).

Example

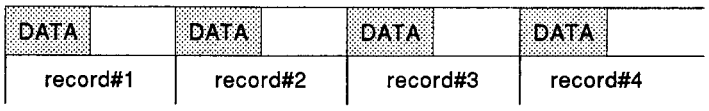


The LOF function returns the value 4.

RANDOM FILE

Data is written in record units, and the length of the file is the highest record number (number of sectors).

Example



The LOF function returns the value 4.

- If the record number specified in the **GET** command is larger than the value returned by the **LOF** function, an **"Input past end"** error occurs.

REMARK

See EOF and LOC. Also see chapter 6.

- Returns the value of the natural logarithm.

LOGarithm

SYNTAX	**LOG (<numeric expression>)**	
	numeric expression	Specifies the value for which the natural logarithm is to be calculated.
EXAMPLE	**A = LOG(10)**	The natural logarithm of 10 is stored in A.

DESCRIPTION

- The **LOG** function returns the natural logarithm of the value specified by numeric expression.
- Calculations are performed in single precision.
- The base 10 logarithm **($\log_{10}X$)** can be calculated using the equation **$\log_{10}X = \log_e X/\log_e 10$**.

PROGRAMMING EXAMPLE

```
10  'Calculate LOGe 100.
20  A=100
30  B=LOG<A>                                  :'Calculate LOGe 100.
40  PRINT LOGe 100 =";B
50  END

RUN
LOGe 100 = 4.60517
OK
```

REMARK

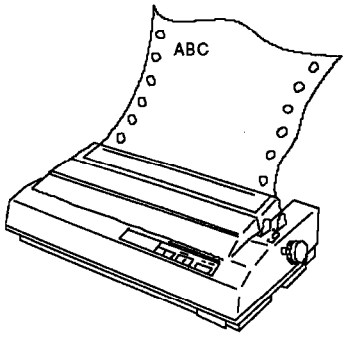
See EXP.

- Outputs data to the printer.

LPRINT

SYNTAX	**LPRINTΔ[<expression>[,<expression>] . . .] [;]**

	expression	Specifies the data to be output to the printer.
EXAMPLE	**LPRINT "ABC"**	"ABC" is output to the printer.



DESCRIPTION

- The **LPRINT** command outputs data to the printer.
- Except for the fact that output is sent to the printer, this command is identical to the **PRINT** command (see the PRINT command).
- The printer to which output is sent is designated with the **ZLDV** command.

POINT

When output is being sent to the printer, all keyboard input including the [Ctrl]+[C] or [Break] keys is disabled. Thus, to abort printer output, it is necessary to turn off the printer power or to set the printer to OFFLINE condition. After about five seconds, the printer output operation is aborted.

REMARK

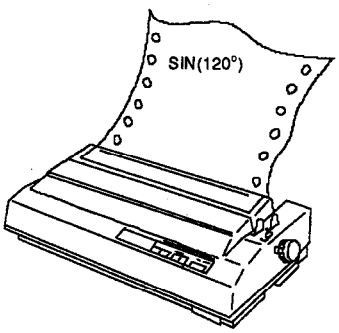
See PRINT, SPC and TAB. Also see section 7.3.3.

- Outputs data to the printer in a specified format.

LPRINT USING

SYNTAX	**LPRINTΔ USINGΔ"<display format>";<expression>[,<expression>] . . . [;]**

	display format	Specifies the format in which the character string or numeric value is to be printed.
	expression	Specifies the character string or numeric value to be printed.
EXAMPLE	**LPRINT USING "SIN(###°)";120**	Prints the value of SIN (120 degrees) to the printer.



DESCRIPTION

- The **LPRINT USING** command outputs data to the printer in a specified format.
- Except for the fact that output is sent to the printer, this command is identical to the **PRINT USING** command (see the PRINT USING command).
- The printer to which output is sent is designated with the **ZLDV** command.

POINTS

(1) When output is being sent to the printer, all keyboard input including the [Ctrl]+[C] or [Break] keys is disabled. Thus, to abort printer output, it is necessary to turn off the printer power or to set the printer to OFFLINE condition. After about five seconds, the printer output operation is aborted.

(2) For information on how to specify the <display format> for the LPRINT USING command, refer to the PRINT USING command.

REMARK

See PRINT USING. Also see section 7.3.3.

- Moves data from memory to the random file buffer in preparation for the PUT command.

Left SET

SYNTAX	**LSETΔ<character string variable>=<character string expression>**

	character string variable	A character variable that was defined with the FIELD command is specified.
	character string expression	Specifies the character string data to be written.
EXAMPLE	**LSET A$="MITSUBISHI"**	The string "MITSUBISHI" is assigned to the variable A$.

LSET RANDOM FILE BUFFER

A$ ⇨

M	I	T	S	U	B	I	S	H	I					

DESCRIPTION

- The **LSET** command moves data from memory to the random file buffer in preparation for the **PUT** command.
- In order to execute this function, a file must first be opened as a random file.
- Before the **LSET** command can be executed, character variables must be defined with the **FIELD** command.
- If the length of the character expression is shorter than the field length assigned to the character variable with the **FIELD** command, the **LSET** command stores the character expression starting from the left side of the field.
  The unused portion of the field is filled in with whitespace.

  Example

  field length : 10
  character expression : NAGOYA

N	A	G	O	Y	A				

field length 10

- If the length of the character expression is longer than the field length assigned to the character variable with the **FIELD** command, the right section of the the character expression is cut off.

  Example

  field length : 10
  character expression : NAGOYASEISAKUSYO

N	A	G	O	Y	A	S	E	I	S

field length 10

- Before a numeric value is moved into the random file buffer, it must be converted into a character string using the **MKI$, MKS$** or **MKD$** functions.

REMARK

See FIELD, PUT, GET and RSET. Also see chapter 6.

- Combines the program currently in memory with a program from a memory card, floppy disk or hard disk.

MERGE

SYNTAX	**MERGEΔ"<drive number>:<system name>\<file name>"**

drive number	Specifies the memory card, floppy disk or hard disk where the program to be combined is stored.
system name	Specifies the system name under which the program to be combined is stored.
file name	Specifies the file name of the program to be combined.

EXAMPLE

LOAD "0:TEST\No1.BAS"

The program currently in memory is combined with the program stored in file "NO1.BAS" under system name "TEST" on the memory card in memory card slot 1.

DESCRIPTION

- The **MERGE** command combines the program currently in memory with a program saved on a memory card, floppy disk or hard disk and stores the new program in memory.
- **<Drive number>** is used to specify the storage device from which to read the program to be combined.<drive number> specifications correspond to physical devices as shown below:

  ON-LINE PROGRAMMING

  AD51H Memory Card 1 0

  AD51H Memory Card 2 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFFLINE PROGRAMMING

  PC/AT FDD 1

  PC/AT HDD[C] 3

- **<System name>** specifies which system name the program to be combined is stored under. If the specified system name does not exist, a "File not found" error occurs.
- **<File name>** specifies the name and file extension of the file in which the program to be combined is stored. A file with no extension can be specified by the file name.

- If lines with the same line number exist in both the program currently in memory and the program to be combined, the lines of the program currently in memory are overwritten by the lines of the program that is read in and combined.
- After the **MERGE** command is executed, BASIC returns to the "waiting for command" condition. If the command is used within a program, it must be used with the **CHAIN** command.

REMAKE

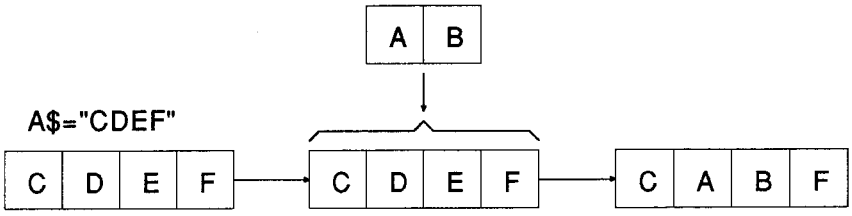
See CHAIN, COMMON, DELETE, LOAD and SAVE. Also see section 3.15.

- Changes part of a character string to another string.

MIDdle $

SYNTAX	**MID$(<character string expression 1>, <interger expression 1> [,<interger expression 2>]) =<character string expression 2>**

	character string expression 1	Specifies the character string or string variable, part of which is to be changed to character string expression 2.
	character string expression 2	Specifies the character string or string variable to be copied into character string expression 1.
	integer expression 1	Specifies the starting position within character string expression 1 of the substring to be changed.
	integer expression 2	Specifies the length of the substring within character string expression 1 to be changed.
EXAMPLE	**MID$(A$,2)="AB"**	Change the substring within A$ starting at the second character to "AB".



DESCRIPTION

- The **MID$ (version 1)** function changes part of a character string to another string.
- If **<integer expression 2>** is omitted, all of **<character string expression 2>** is copied to <character string expression 1>. However, even if **<integer expression 2>** is not specified, the character string cannot be copied past the end of **<character string expression 1>**.
- The value specified for **<integer expression 1>** cannot be larger than the number of characters in **<character expression 1>**. Also, values lower than zero cannot be specified.
- The empty string cannot be specified for **<character string expression 1>**.

PROGRAMMING EXAMPLE

```
10  'Copy a character string
20  A$=f"ABCDEFGHI"                          :'Define a character string
30  B$="XYZ"                                 :'Define the character string to be copied
40  PRINT "The first string---------->";A$
50  MID$(a$,3,3)k=B$                         :'Copy
60  PRINT "The string after copying-->";A$
70  END

RUN
The first string---------->ABCDEFGHI
The string after copying-->ABXYZFGHI
OK
```

REMARK

See MID$ (version 2). Also see section 3.13.

- Returns the substring starting at a specified position within a character string.

MIDdle $

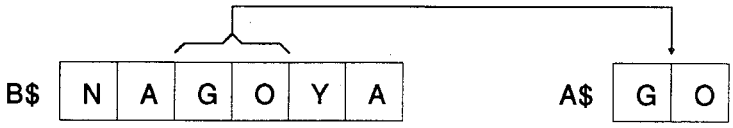
SYNTAX	**MID$(<character string expression>,< integer expression 1> [,<integer expression 2>])**

character string expression	Specifies the character string or string variable, from which the substring is to be read.
integer expression 1	Specifies the starting position within character string expression 1 of the substring to be read.
integer expression 2	Specifies the length of the substring within character string expression 1 to be read.

EXAMPLE

A$=MIDD$(B$,3,2) Reads the third and the fourth character from the left in B$, stores them to A$.

B$="NAGOYA"



DESCRIPTION

- The **MID$ (version 2)** function reads out and returns the substring starting at a specified position within a character string.
- If **<integer expression 2>** is omitted, or if the number of characters between the starting position specified with **<integer expression 1>** and the end of the string is less than **<integer expression 2>**, the function returns a substring consisting of all of the characters from the position specified with **<integer expression 1>** to the end of the string.
- If the value of **<integer expression 1>** is larger than the length of **<character string expression>**, the function returns the empty string.

PROGRAMMING EXAMPLE

```
10  'Read a substring
20  A$=AD51H-BASIC"                    :'Define a character string
40  B$=MID$(A$,7,5)                    :'Read out five characters staring with the
                                        seventh character
50  PRINT "Substring-->";B$
60  END

RUN
Substring-->BASIC
OK
```

REMARK

See RIGHT$, LEFT$, and MID$ (version 1). Also see section 3.13.

- Converts a double precision real number value into a character string.

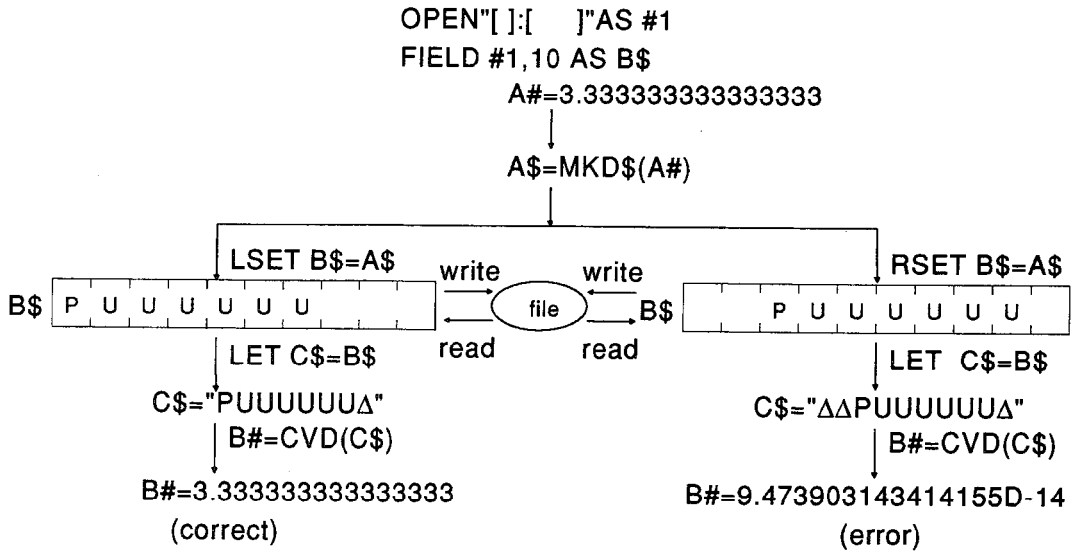
MaKe Double $

SYNTAX	**MKD$ (<double precision expression> [,S])**	
	double precision expression	Specifies the double precision real number to be converted into a character string.
EXAMPLE	**C$=MKD$(3.14159265389)**	The number is converted to an eight byte character string and stored to C$.

DESCRIPTION

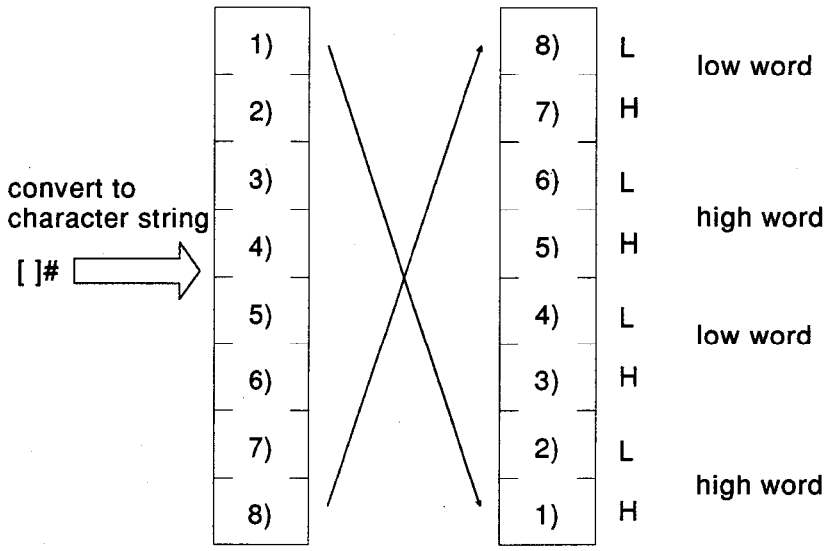
- The **MKD$ function** converts a double precision real format numeric value into an eight byte character string.
- The **MKD$** function must be used to convert double precision numeric values to character strings before the values are written to the random file buffer with the **LSET** or **RSET** commands.
- Once a numeric value has been converted to a character string with the **MKD$** function, only the **CVD** function can be used to reconvert the string back to the original numeric value.
- It is recommended that the **LSET** command be used to write a string converted with the **MKD$** function to the random file buffer.
  If the length (number of bytes) of the field set with the **FIELD** command is longer than the length (number of bytes) of the string converted with the **MKD$** function, if the **RSET** command is used to write the string to the field, when the string is reconverted with the **CVD** function, the results are incorrect.

  Example



- The **CVD** function converts only the required number of bytes starting from the left of the variable area declared with the **FIELD** command.

- Data converted with the MKD$ function can be used for data communication as well as for random files.
- If the **[,S]** option is selected, the double precision real number is converted into a character string, but the character string is stored as shown below.



REMARK

See MKI$, MKS$, CVD, CVI and CVS.

- Converts data from IEEE double precision internal representation to a string that can be converted into a double precision real number using the CVD function.

MaKe Double Microsoft Binary Format $

SYNTAX	**MKDMBF$ (<numeric expression>)**

	numeric expression	Specifies a numeric value represented using IEEE double precision internal representation.
EXAMPLE	**A#=CVD(MKDMBF$(B#))**	The value of B#, which is represented in IEEE double precision internal representation, is converted to AD51H-BASIC double precision internal representation and stored to A#.

DESCRIPTION

- The **MKDMBF$** function converts data from IEEE double precision internal representation to a string that can be converted into a double precision real number using the AD51H-BASIC function CVD.
- The IEEE internal representation of double precision values and the AD51H-BASIC internal representation of real numbers are different. The **MKDMBF$** function is used to convert between the two representations.
- The value specified for **<numeric expression>** must be a value represented by IEEE double precision internal representation. If other values are specified, proper conversion cannot take place.
- The **MKDMBF$** function can convert only double precision real number data.
- The **MKDMBF$** function is used to convert data from a system that used the IEEE internal representation for double precision real numbers.

REMARK

See CVDMBF, CVSMBF and MKSMBF$.

- Converts an integer value into a character string.

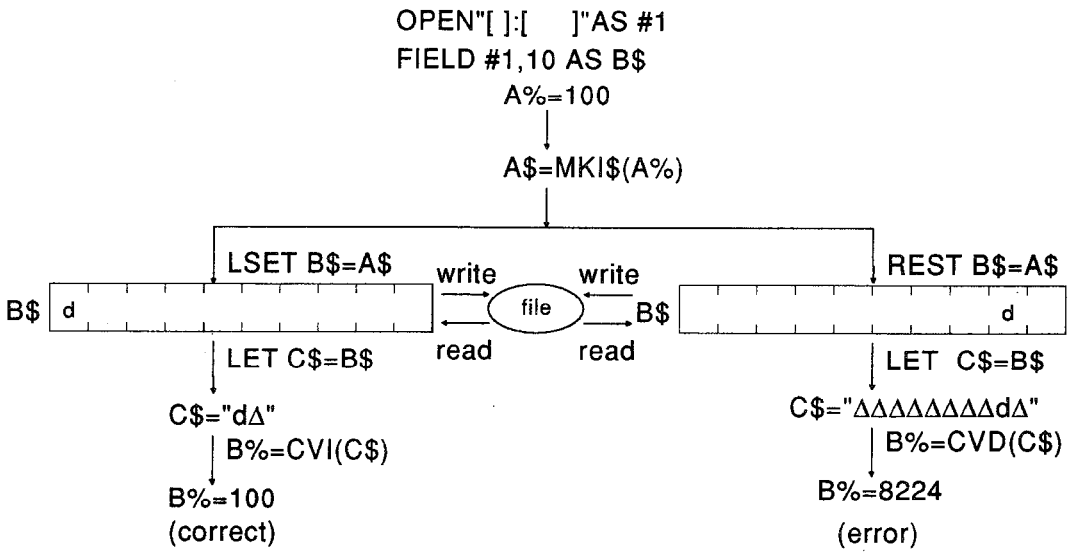
MaKe integer $

SYNTAX	**MKI$ (<integer expression> [,S])**	
	integer expression	Specifies the integer to be converted into a character string.
EXAMPLE	**A$=MKI$(314)**	The number is converted to a two byte character string and stored to A$.

DESCRIPTION

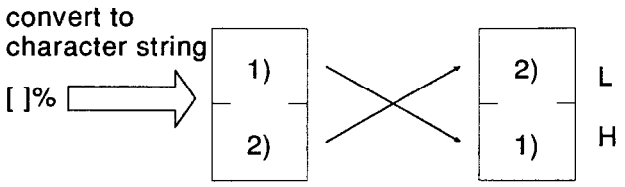
- The **MKI$** function converts an integer format numeric value into a two byte character string.
- The **MKI$** function must be used to convert integer values to character strings before the values are written to the random file buffer with the **LSET** or **RSET** commands.
- Once a numeric value has been converted to a character string with the **MKI$** function, only the **CVI** function can be used to reconvert the string back to the original numeric value.
- It is recommended that the **LSET** command be used to write a string converted with the **MKI$** function to the random file buffer.
  If the length (number of bytes) of the field set with the **FIELD** command is longer than the length (number of bytes) of the string converted with the **MKI$** function, if the RSET command is used to write the string to the field, when the string is reconverted with the **CVI** function, the results are incorrect.

  Example



The **CVI** function converts only the required number of bytes starting from the left of the variable area declared with the **FIELD** command.

- Data converted with the **MKI$** function can be used for data communication as well as for random files.
- If the **[,S]** option is selected, the integer is converted into a character string, but the character string is stored as shown below.



REMARK

See MKD$, MKS$, CVD, CVI and CVS.

- Converts a single precision real number value into a character string.

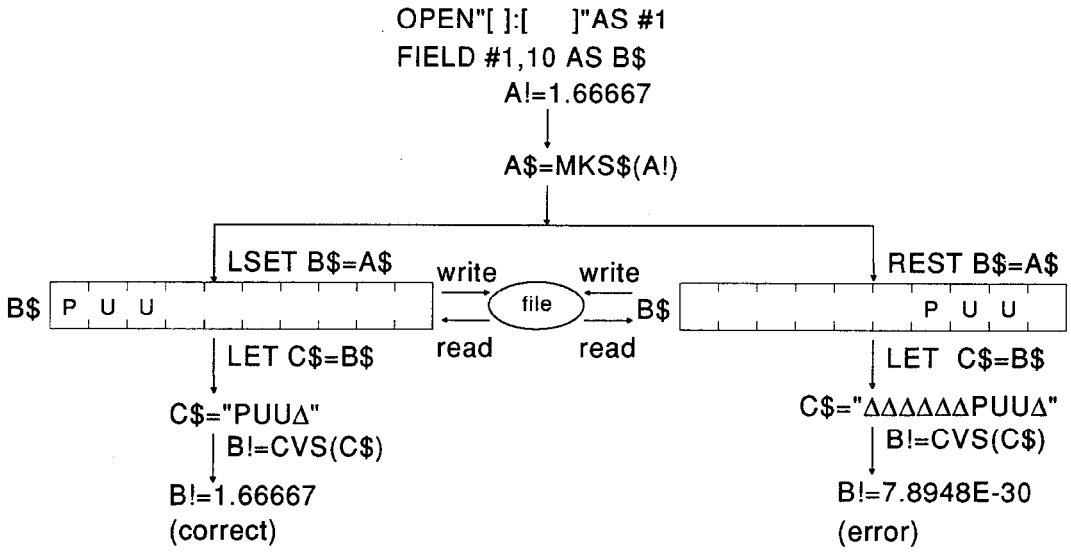
MaKe Single $

SYNTAX	**MKS$ (<single precision expression> [,S])**	
	single precision	Specifies the single precision real number to be converted into expression a character string.
EXAMPLE	**B$=MKS$(3.14159)**	The number is converted to a four byte character string and stored to B$.

DESCRIPTION

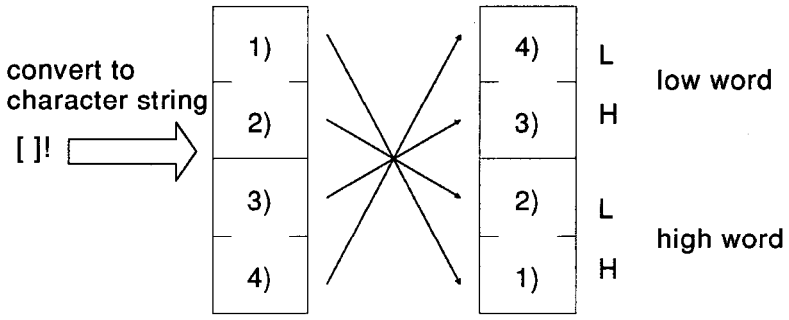
- The **MKS$** function converts a single precision real format numeric value into a four byte character string.
- The **MKS$** function must be used to convert single precision numeric values to character strings before the values are written to the random file buffer with the **LSET** or **RSET** commands.
- Once a numeric value has been converted to a character string with the **MKS$** function, only the **CVS** function can be used to reconvert the string back to the original numeric value.
- It is recommended that the **LSET** command be used to write a string converted with the **MKS$** function to the random file buffer.
  If the length (number of bytes) of the field set with the **FIELD** command is longer than the length (number of bytes) of the string converted with the **MKS$** function, if the **RSET** command is used to write the string to the field, when the string is reconverted with the CVS function, the results are incorrect.

Example



The **CVS** function converts only the required number of bytes starting from the left of the variable area declared with the **FIELD** command.

- Data converted with the **MKS$** function can be used for data communication as well as for random files.
- If the **[,S]** option is selected, the single precision real number is converted into a character string, but the character string is stored as shown below.



REMARK

See MKD$, MKI$, CVD, CVI and CVS.

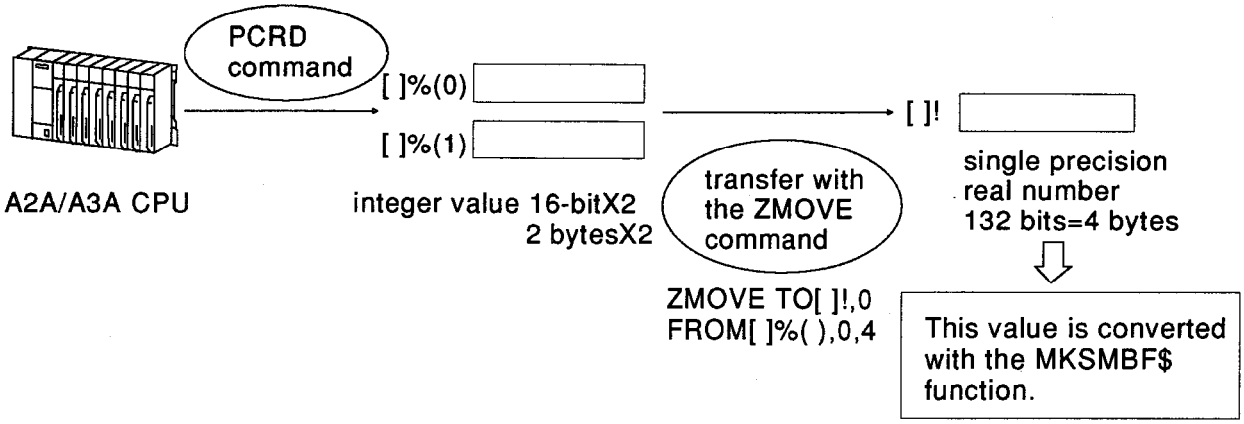
- Converts data from the internal representation of floating point values used by A2A and A3A CPUs (IEEE single precision internal representation) to a string that can be converted into a single precision real number using the CVS function.

MaKe Single Microsoft Binary Format $

SYNTAX	**MKSMBF$ (<numeric expression>)**	
	numeric expression	Specifies a numeric value represented in the floating point internal representation used by A2A and A3A CPUs(IEEE single precision internal representation).
EXAMPLE	**A!=CVS(MKSMBF$(B!))**	The value of B!, which is represented in the floating point internal representation used by A2A and A3A CPUs (IEEE single precision internal representation), is converted to AD51H-BASIC single precision internal representation and stored to A!.

DESCRIPTION

- The **MKSMBF$** function converts data from the internal representation of floating point values used by A2A and A3A CPUs (IEEE single precision internal representation), to a string that can be converted into a single precision real number using the AD51H-BASIC function **CVS**.
- The internal representation of floating point values used by A2A and A3A CPUs (IEEE single precision internal representation) and the AD51H-BASIC internal representation of real numbers are different. The **MKSMBF$** function is used to convert between the two representations.
- The value specified for **<number expression>** must be a value represented in the floating point internal representation used by A2A and A3A CPUs (IEEE single precision internal representation). If other values are specified, proper conversion cannot take place.
- The **MKSMBF$** function can convert only single precision real number data.
- When reading floating point real number values from an A2A or A3A CPU using the PCRD command, the data should be stored in an integer array. Once this is done, the **ZMOVE** command can be used to store the data in a single precision real number variable, and then the **MKSMBF$** function can be used to convert the data.



- The **MKSMBF$** function is not only used for converting floating point data from A2A and A3A CPUs, but is is also used for converting data from systems that use the IEEE internal representation for single precision real numbers.

PROGRAMMING EXAMPLE

```
10  'Read a floating point real number from an AnA CPU and convert it to a single preci
    sion real number.
30  DIM TBL%(5),A%(3)                          :'Define arrays
40  TBL%(0)=255                                :'Set the communication station to the host
50  TBL%(1)=1                                  :'Select read device memory data
60  TBL%(2)=2                                  :'Select word units
70  TBL%(3)=18                                 :'Select data register
80  TBL%(4)=0                                  :'Specify the head device number
90  TBL%(5)=2                                  :'Specify the number of points to read
100  PCRD TBL%( ),A%( )                        :'Execute read
105  A!=0                                      :'Store a dummy value to A!
110  ZMOVE TO A!,0 FROM A%( ),0,4              :'Store the data that was read to A!
120  B!=CVS(MKSMBF$(A!))                       :'Execute conversion
130  PRINT "The floating point real number in D0, D1 is-->";B!
140  END
```

REMARK

See CVDMBF, CVSMBF and MKDMBF$.

- Changes the name of a program or data file stored on a memory card, floppy disk or hard disk.

NAME

SYNTAX	**NAMEΔ"<drive number>:<system name\><old file name>** **"ΔASΔ"<drive number>:<system name\><new file name>"**

drive number	Specifies the memory card, floppy disk or hard disk where the file whose name is to be changed is stored.
system name	Specifies the system name under which the file whose name is to be changed is stored.
old file name	Specifies the name of the file whose file name is to be changed.
new file name	Specifies the new file name to be assigned to for the file.

EXAMPLE

NAME "0:TEST\No1.BAS"AS"0:TEST\No2.BAS"

The name of the file "NO1.BAS" stored under system name "TEST" on the memory card in memory card slot 1 is changed to "NO2.BAS".

DESCRIPTION

- The **NAME** command changes the name of a program or data file stored on a memory card, floppy disk or hard disk.
- **<Drive number>** and **<system name>** must be the same for both **<old file name>** and **<new file name>**.
- **<Drive number>** specifies the memory card, floppy disk or hard disk where the file whose name is to be changed is stored. The values for <drive number> correspond to physical devices as shown in the table below:

  ON-LINE PROGRAMMING

  AD51H Memory Card 1 0

  AD51H Memory Card 2 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFFLINE PROGRAMMING

  PC/AT FDD 1

  PC/AT HDD[C] 3

- **<System name>** specifies which system name under which the file whose name is to be changed is stored. If the **<system name>** is not specified, the file is assumed to have no system name.

- **<Old file name>** specifies the name and file extension of the file whose name is to be changed. If the file extension is omitted, the new file name will have no extension.
- **<New file name>** specifies the new name and file extension to be assigned to the file. If the file extension is omitted, the new file name will have no extension.
- The **NAME** command cannot be execute when a file is open. If a file is open and the **NAME** command is executed, a **"File already open"** error occurs.

REMARK

See section 3.4.2.

- Deletes the program currently in memory and initializes all variables.

NEW

SYNTAX	**NEW**	
EXAMPLE	**NEW**	The program currently in memory is deleted, and all variables are initialized.

DESCRIPTION

- The **NEW** command clears the current program and initializes memory in preparation for entering a new program. However, the sizes of memory areas defined with the **CLEAR** command do not change.

- After the **NEW** command is executed, BASIC enters the "waiting for command input" condition.

- If there are any open files when the **NEW** command is executed, they are allautomatically closed.

REMARK

See CLOSE, END, LOAD and SAVE. Also see section 3.1.

- Converts a numeric value to an octal (base 8) number stored as a character string.

OCTal $

SYNTAX	**OCT $(<nuberic expression>)**	
	numeric expression	Values in the range -32768 to 65535 can be specified. * Values in the range 32768 to 65535 return the same results as values in the range -32768 to -1 (8000_H to $FFFF_H$).
EXAMPLE	**PRINT OCT$(10)**	The octal representation of the decimal (base 10) number 10 is displayed.

DESCRIPTION

- The **OCT$** function converts a numeric value into an octal (base 8) number represented as a character string.
- The decimal section of **<numeric value>** is truncated to convert it to an integer before it is converted to an octal number.
- A conversion table for decimal (base 10) and octal (base 8) numbers is included below.

octal	0	1	2	3	4	5	6	7	10	11	12
decimal	0	1	2	3	4	5	6	7	8	9	10

PROGRAMMING EXAMPLE

```
10  'Convert a decimal (base 10) number to an octal number.
20  A=100                                   :'Define the number
30  A$=OCT$(A)                              :'Convert to octal
40 PRINT "decimal =";A
50 PRINT " octal =";A$
60 END

RUN
decimal= 100
 octal=144
OK
```

REMARK

See VAL, BIN$ and HEX$.

Octal numbers are indicated by the characters "&O" before the number.

Example

decimal		octal
200	⟶	&O310

The VAL function is used to convert an octal number represented as a character string into a decimal number.

Example

		decimal
VAL ("&O310")	⟶	200

↓

"&O" must be added at the beginning.

- Defines the starting line of the control routine to which execution is branched when the data is received from the communication line.

ON COMmunication
GO to SUBroutine

SYNTAX	**ONΔCOM(<channel number>) GOSUB <line number>**	
	channel number	Specifies the communication line where data will be received.
	line number	Specifies the line number to be branched to when data is received.
EXAMPLE	**ON COM (1) GOSUB 1000**	If input is received from the CH1 (RS-232C) communication line, execution is branched to line 1000.

DESCRIPTION

- The **ON COM GOSUB** command defines the starting line of the control routine to which exection is branched when data is received from the communication line.

- The following values may be specified for **<channel number>**.

  1 ..... CH1 (RS-232C)

  2 ..... CH2 (RS-232C)

  3 ..... CH3 (RS-422)

- For input from a keyboard attached to the AD51H, the **ON KEY GOUSUB** command should be used.

- When execution is paused awaiting input due to execution of an **INPUT** or similar command, if data is received from the communication line, the control routine is not executed until execution of the current command has completed.

- Returning from a control routine is accomplished in the same way as with regular subroutines using the **RETURN** command. If only the **RETURN** command is entered, execution returns to the command following the **ON COM GOSUB** command; if a line number is specified following the RETURN command, execution is returned to the specified line.

PROGRAMMING EXAMPLE
(executed in AD51H BASIC)

```
10  'Only when data is received at channel 2 is it accepted.
20  ON ERROR GOTO 500                        :'If there is an error, branch to line 500
30  CLS
40  ON COM(2) GOSUB 340                      :'If there is data received at CH2, branch
                                              to line 340
50  COM(2) ON                                :'Enable data to be received
60  DIM TBL1%(2),CNT%(10),TBL2%(2)
70  '***** RS-232C CH.OPEN *****
80  CH%=2                                    :'Define the communication channel
90  TBL1%(0)=4800                            :'Set the baud rate
100  TBL1%(1)=&H8                            :'Set the character length and parity bit
110  TBL1%(2)=&H1                            :'Set the stop bit
120  ZOPEN #CH%,TBL1%()                      :'Open the communication channel
130  '
140  '***** PORT CNTL *****
150  CNT%(0)=32                              :'Set RS and ER signal control
160  CNT%(1)=&H1                             :'Set the RS signal ON
170  ZCNTL #CH%,0,CNT%()                     :'Execute
180  '
190  '***** MAIN PROGRAM *****
200  J=0                                     :'Screen display
210  LOCATE 0,J
220  PRINT "MAIN PROGRAM"
230  J=J+1
240  IF J=10 THEN J=0
250  FOR I=0 TO 100
260  NEXT I
270  CLS
280  GOTO 210
290  '
300  '***** RS-232C CH.CLOSE *****
310  ZCLOSE #CH%
320  END
330  '
340  '***** DATA RECEIVE *****
350  TBL2%(0)=12                             :'Set the required number of bytes to send
360  TBL2%(1)=0                              :'Store the number of bytes to send
370  TBL2%(2)=300                            :'Set the timeout period
380  WOR%=TBL2%(0)/2
390  DIM CTBL%(WOR%-1)
400  ZRECEIVE #CH%,0,TBL2%(),CTBL%()         :'Receive data
410  LOCATE 0,15
420  PRINT "RECEIVE CHARACTER =";TBL2%(1)    :'Display the number of characters received
430  FOR I%=0 TO WOR%-1
440  LOCATE 0,16+I%
450  PRINT "RECEIVE DATA =&H";HEX$(CTBL%(I%)):'Display the data received
460  NET I%
470  RETURN 310
480  '
490  '***** ERROR ROUTINE *****
500  ERTYPE%=ERR:ERLINE%=ERL                 :'Store the error and the line where it
                                              occurred
510  IF ERTYPE%=17 THEN PRINT "ZOPEN ERROR!!":'If it was a Z-type error, display
520  PRINT "ERROR CODE =";ERTYPE%            :'Display the error code
530  PRINT "ERROR LINE =";ERLINE%            :'Display the error type
540  RESUME 310                              :'Return control to line 310
```

- Enables the procedure to be executed when an error occurs, and defines the starting line of the error management program.

ON ERROR GOTO

SYNTAX	**ONΔERRORΔGOTOΔ<line number>**	
	line number	Specifies the starting line number or label of the error management program that is executed if an error occurs.
EXAMPLE	**ON ERROR GOTO 100**	If an error occurs, execution is shifted to line 100.

DESCRIPTION

- The **ON ERROR GOTO** command enables the procedure to be executed when an error occurs and defines the starting line of the error management program.
- A label can be used instead of an actual line number for **<line number>.**
- If the specified **<line number>** does not exist, an **"Undefined line number"** error occurs.
- The error management routine should not perform character display, nor should it change I/O ports with the **ZODV** or **ZIDV** commands.
  Depending on the timing of the execution of the error management program, sometimes it is not possible to display characters normally or perform normal input/output when execution is returned to the main program.
- To disable execution of the error management program, zero is specified for **<line number>**.
  If this is performed, when an error occurs, the error message is displayed and program execution is aborted.
- If the **ON ERROR GOTO 0** command is executed within the error management program,the error message is displayed and program execution is aborted.
- If an error is detected within the error management program that cannot be corrected, it is best to execute the **ON ERROR GOTO 0** command.
- The **RESUME** command is used to return from the error management program.
- If an error occurs during execution of the error management program, the BASIC error message is displayed, and program execution is aborted.

PROGRAMMING EXAMPLE

```
10 'Display "ERROR" when an error occurs.
20  ON ERROR GOTO 50                        :'Define the location to branch to if an
                                             error occurs
30  ERROR 10                                :'Cause an error
40  END
50  PRINT "ERROR"
60  RESUME 40                               :'Error management is complete

RUN
ERROR
OK
```

REMARK

See ERROR, RESUME, ERL and ERR. Also see chapter 9.

- Branches execution to one of several specified line numbers based on a value.

ON~GOSUB

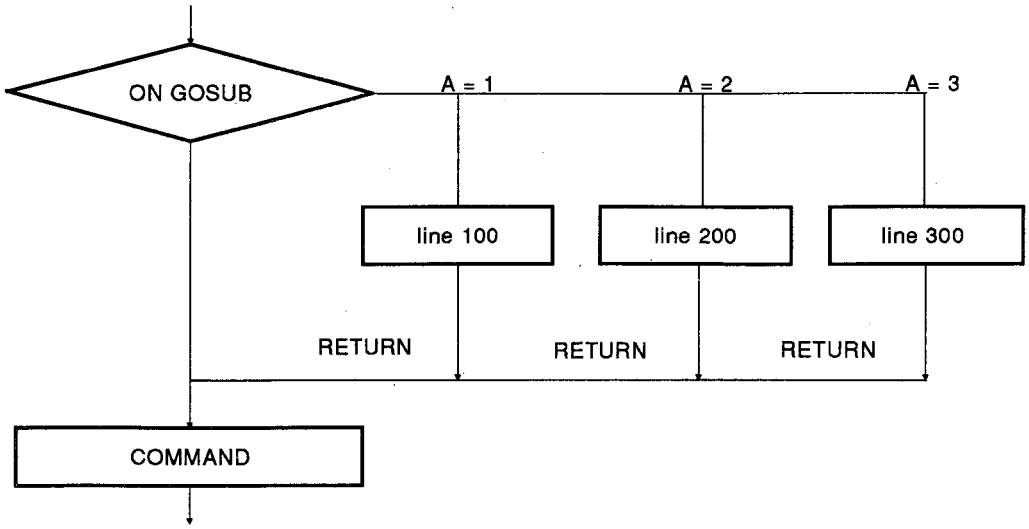
SYNTAX	**ONΔ<expression>ΔGOSUBΔ<line number> [, <line number> . . .]**

	expression	Specifies the expression whose value will determine to which line number execution will be branched.
	line number	Specifies the line number to which execution will be branched.
EXAMPLE	**ON A GOSUB 200,300,400**	If the value of A is 1, the subroutine starting at line 200 is executed; if A is 2, the subroutine starting at line 300 is executed; and if A is 3, the subroutine starting at line 400 is executed.

DESCRIPTION

- The **ON~GOSUB command** branches execution to one of several specified line numbers based on a value.
- If the value of **<expression>** is not an integer, the decimal portion is truncated.
- If the value of **<expression>** is zero or if it is greater than the total number of line numbers specified (up to 255 may be specified), execution falls to the next command.
- If the value of **<expression>** is negative, or if it is greater than 255, an **"Illegal function call"** error occurs.
- A label can be specified in place of an actual line number for **<line number>**.
- Each line number specified in the **ON~GOSUB** command should be the starting line of a procedure.
- If a command is written following the **ON~GOSUB** command, after the procedure is executed and execution is returns with the **RETURN** command, the command following the **ON~GOSUB** command is executed.

```
ON[A]GOSUB[100], [200], [300] : [COMMAND]
```



PROGRAMMING EXAMPLE

```
10  'Branch to a subroutine depending on the value of variable A
20  INPUT A                                  :'Input the value
30  ON A GOSUB 100,200,300                   :'Branch to a subroutine depending on the value
40  END
100  PRINT "This subroutine is executed when 1 is entered." : RETURN
200  PRINT "This subroutine is executed when 2 is entered." : RETURN
300  PRINT "This subroutine is executed when 3 is entered." : RETURN

RUN
? 1
This subroutine is executed when 1 is entered.
OK
RUN
? 2
This subroutine is executed when 2 is entered.
OK
RUN
? 3
This subroutine is executed when 3 is entered.
OK
RUN
? 4
OK
```

REMARK

See GOSUB.

- Branches execution to one of several specified line numbers based on a value.

ON~GOTO

SYNTAX	**ONΔ<expression>ΔGOTOΔ<line number> [,<line number>...]**

	expression	Specifies the expression whose value will determine to which line number execution will be branched.
	line number	Specifies the line number to which execution will be branched.
EXAMPLE	**ON A GOTO 200,300,400**	If the value of A is 1, execution is branched to 200; if A is 2, execution is branched to line 300; and if A is 3, execution is branched to line 400.

DESCRIPTION

- The **ON~GOTO** command branches execution to one of several specified line numbers based on a value.
- If the value if **<expression>** is not an integer, the decimal portion is truncated.
- If the value of **<expression>** is zero or if it is greater than the total number of line numbers specified (up to 255 may be specified), execution falls to the next command.
- If the value of **<expression>** is negative, or if it is greater than 255, an **"Illegal function call"** error occurs.
- A label can be specified in place of an actual line number for **<line number>.**

PROGRAMMING EXAMPLE

```
10 'Branch depending on the value of variable A
20 INPUT A                                    :'Input the value
30 ON A GOTO 100,200,300                      :'Branch depending on the value
40 END
100 PRINT "This procedure is executed when 1 is entered.":END
200 PRINT "This procedure is executed when 2 is entered.":END
300 PRINT "This procedure is executed when 3 is entered.":END

RUN
? 1
This procedure is executed when 1 is entered.
OK

RUN
? 2
This procedure is executed when 2 is entered.
OK
```

```
RUN
? 3
This procedure is executed when 3 is entered.
OK
RUN
? 4
OK
```

REMARK

See GOTO. Also see section 3.6.

- Opens a file and enables I/O processing.

OPEN

SYNTAX	**OPEN Δ"[<drive number>:] [<system name>\] <file name>" [ΔFORΔ<mode>]ΔASΔ[#] <file number>**

drive number	Specifies the memory card, floppy disk or hard disk where the file to be opened is located.
system name	Specifies the system name where the file to be opened is located.
file name	Specifies the name of the file to be opened.
mode	For sequential files, specifies whether an input, output or append procedure is to be performed with the file.
file number	Specifies a file number (in the range 1 - 8) for the file to be created.

EXAMPLE

OPEN"1:ABCD" FOR OUTPUT AS #1

The file named "ABCD" on the memory card [2] drive is opened asfile number 1 for output processing.

DESCRIPTION

- The **OPEN** command opens a file and enables I/O processing.
- The file specified with **<system name>** and **<file name>** is assigned an I/O buffer and a **<file number>**, and is defined for either input or output processing.Following this, to perform input or output of data with this file, the file number is used.
- Input/output to a file will be executed if the file has been opened by the OPEN command.
- The values that can be specified for **<drive number>** correspond to physical devices as shown below:

  AD51H Memory Card [1] 0

  AD51H Memory Card [2] 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

- If any drive number other than those listed above are specified, a **"Bad drive number"** error occurs.

- Processing must be specified as either input or output for the file. One of the following should be selected:

  INPUT: Input processing is selected for a sequential file.

  OUTPUT: Output processing is selected for a sequential file.

  APPEND: Data is added to the end of an existing file.

- If **FOR <mode>** is omitted, the file is assumed to be a random file, and both input and output can occur without prior specification.
- If **INPUT** or **APPEND** are selected, but the specified file does not exist, a **"File not found"** error occurs.
- Open files are automatically closed if the program is edited, or whenever any of the following commands are executed:

  RUN, CLEAR, END, NEW, LOAD, CLOSE

POINT

If OUTPUT mode is specified for a file that already exists, all data that is recorded in the file is deleted.

REMARK

See chapter 6.

- Reads out all types of data from the PC CPU.

PC ReaD

SYNTAX	**PCRDΔ<control table>, <data storage area>**	
	control table	Specifies the type of data to be read from the PC CPU.
	data storage area	Specifies the variable in which to store the data that is read from the PC CPU.
EXAMPLE	**PCRD A%( ),B%( )**	The data specified in integer array variable A% is read from the PC CPU and stored in integer array variable B%.

DESCRIPTION

- The PCRD command reads out all types of data from the PC CPU.
- The details of the procedure are designated with procedure codes in the control table. The procedure details correspond to the procedure codes as follows:

Procedure Code	Procedure Details
1	Read device memory
2	Monitor a device that was recorded for monitoring with the PCWT command
4	Read an extension file register
5	Monitor an extension file register that was recorded for monitoring with the PCWT command
7*	Read from consecutive addresses in the extension file register
8	Read the sequence program
9	Read the microcomputer program
10	Read comment data
11	Read extension comment data
12	Read the special function module's buffer memory
13	Read the type name of the PC CPU
14	Read parameter data

*Procedure code 7 (read from consecutive addresses in the extension file register, is only valid for A2A and A3A CPUs.

POINT

Although the <control table> uses less than 10 array elements, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the PCRD command is executed, an error occurs.
(Usually, if fewer than ten elements are required, it is not necessary to define an array with the DIM command.)

- The **<data storage area>** specifies where the data read from the PC CPU is to be stored. integer variables, integer arrays, character variables, character arrays, or the internal devices ED and EM can be used. The <data storage area> is specified as follows:

integer variable	[ ]%	
character variable	[ ]$	
integer array	[ ]%( )	
character array	[ ]$(n)	
ED	W@(ED,n)	Indicated by the special
EM	B@(EM,n)	

POINT

Before executing the PCRD command, the variable to be used as the data storage area must be filled with dummy data. If dummy data is not stored in the variable, when the PCRD command is executed, an error occurs.

integer variables [ ]% = 0

character variables [ ]$ = SPACE$ (255)

For ED, and EM, it is not necessary to set dummy data.

If an array is used for the <control table>, even though less than 10 array elements are used, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the PCRD command is executed, an error occurs.

- For information on how data is stored, see section 4.2.6.
- All procedures that are performed when the PCRD command is executed may be executed even when the PC CPU is running.

Procedure Code 1 **Read device memory data**

Control table components

Elememt Position	Item	details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	units to process	Specifies with what units to process the bit device
[ ]%(3)	device code	Specifies the device code(a device symbol is specified)
[ ]%(4)	device number	Specifies the device number
[ ]%(5)	number of points	Specifies the number of points of data to read.

- Data can be read from all device memory with the exception of the PC CPU extension file register.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.

- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For detailes see section 4.2.

[ ]%(1) Procedure code

- 1 is specified.

[ ]%(2) When reading from a bit device, specifies whether to perform the operation in bit units or word units.

- To specify bit units 1
- To specify word units 2

When reading from a word device, 2 must be specified.

[ ]%(3) Specifies the symbol that has been decided for the device code.
For information on the code numbers, see section 4.2.5.

[ ]%(4) Specifies the device number for the device selected by [ ]%(3).
For the range of device numbers that can be specified, see section 4.2.5.
However, when word unit handling has been selected for a bit device, the device number must be specified as zero or a multiple of 16.

[ ]%(5) Specifies how many points of the device are to be read (including the device specified by [ ]%(3) and [ ]%(4)). The range of acceptable values are as follows:

When bit unit handling is selected 1 to 256 points

When word unit handling is selected

- Bit devices 1 to 32 points (words)
- Word devices 1 to 64 points

- If the file register (R) is specified as the read device, the following conditionsapply:
  (1) If the block number of the file register has not been changed at all by the PC CPU, the data in block number zero of the file register is read.
  (2) If the block number of the file register has been changed by the PC CPU, the data in the block number that was changed is read.
    - The block number can be changed by the RSET command, which is included in the Microcomputer Program Package of the SW[ ]GHP-UTLPC-FN1 UtilitiesPackage.
    - The block number can also be changed with the RSET command for use only by A2A and A3A CPUs.

POINT

If an A2A or A3A CPU is in use, and the RSET command is used to set the file register block number to a value greater than or equal to 29, procedure code 1 cannot be used. Instead, procdur code 4 (read data from the extension file register) should be used.

When the extended file register is used, in order to be sure that the process is carried out correctly, **procedure code 4** (read data from the extension file register) should be used to read data from the file register.

PROGRAMMING EXAMPLE

```
10 'Read from the PC CPU device memory M0 to M9
20 DIM TBL%(5),A%(9)                          :'Define arrays
30 TBL%(0)=255                                :'Set the communication station number to
                                               the host
40 TBL%(1)=1                                  :'Select "read from device memory"
50 TBL%(2)=1                                  :'Select bit units
60 TBL%(3)=3                                  :'Select internal relay (M)
70 TBL%(4)=0                                  :'Specify the head device number (0)
80 TBL%(5)=10                                 :'Specify the number of points to read
90 PCRD TBL%( ),A%( )                         :'Execute read
100 FOR I=0 TO 9                              :'Display the data that was read
110  PRINT "M";I;
120  IF A%(I)=1 THEN PRINT "=ON " ELSE PRINT "=OFF"
130 NEXT I
140 END
```

Procedure Code 2 **Read data from a device that was recorded for monitoring with the PCWT command**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

- A device that was recorded for monitoring with the **PCWT** command can be monitored, and the results can be read out. If data is read out before the device is recorded for monitoring, an error occurs.

- The following data is specified with the **<control table>:**

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/Ostation when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 2 is specified.

- If the file register (R) is specified as the device to monitor, the following conditions apply:
  (1) If the block number of the file register has not been changed at all by the PC CPU, the data in block number zero of the file register is monitored.
  (2) If the block number of the file register has been changed by the PC CPU, the data in the block number that was changed is monitored.
    - The block number can be changed by the RSET command, which is included in the Microcomputer Program Package of the SW[ ]GHP-UTLPC-FN1 Utilities Package.
    - The block number can also be changed with the RSET command for use only by A2A and A3A CPUs.

POINT

If an A2A or A3A CPU is in use, and the RSET command is used to set the file register block number to a value greater than or equal to 29, procedure code 2 cannot be used. Instead, procedure code 5 (monitor data from an extension file register that was recorded for monitoring with the PCWT command) should be used.

When the extension file register is used, in order to be sure that the process is carried out correctly, **procedure code 5** (monitor data from an extension file register that was recorded for monitoing with the PCWT command) should be used toread data from the file register.

PROGRAMMING EXAMPLE

```
200 DIM TBL2%(1),B%(4)                          :'Define arrays
210 TBL2%(0)=255                                :'Set the communication station number
                                                 to the host
220 TBL2%(1)=2                                  :'Specify "monitor a device recorded for
                                                 monitoring with the PCWT command"
230 PCRD TBL2%( ),B%( )                         :'Execute monitoring
240 PRINT "The condition of X000-X015 =&H";B%(0) :'Display
250 PRINT "The condition of X016-X031 =&H";B%(1)
260 PRINT "The condition of M096-M113 =&H";B%(2)
270 PRINT "The condition of T000 (connection) =&H";B%(3)
280 PRINT "The contents of the D048 settings =";B%(4)
```

Procedure Code 4

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	block number	Specifies the extension file register block number
[ ]%(3)	device number	Specifies the extension file register device number
[ ]%(4)	number of points	Specifies the number of points of data to read

- Data can be read from the extension file register (all block numbers greater than or equal to 1 within the file register).

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 4 is specified.

[ ]%(2) Specifies the extension file register block number.
- If zero is specified for the block number, the PC CPU file register data is read. (This is the same as if device R[ ] is specified for "read device memory.")
- If the value specified for the block number is one or greater, extension file register data is read.
- The largest value that can be specified for the block number depends on the memory cassette installed in the PC CPU. For details, see the ACPU Programming Manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions), and the SW[ ] GHP-UTLP-FN1 Operating Manual.

[ ]%(3) Specifies the device number of the file register (R). The largest value that can be specified is defined by a PC CPU parameter.

[ ]%(4) Specifies how many points of the device specified in [ ]%(3) are to be read. The range of acceptable values is 1 to 64.

PROGRAMMING EXAMPLE

```
10 'Read data from the extension file register (block number 1 R0)
20 DIM TBL%(4),A%(9)                          :'Define arrays
30 TBL%(0)=255                                :'Set the communication station number to
                                               the host
40 TBL%(1)=4                                  :'Select "read from the extension file
                                               register"
50 TBL%(2)=1                                  :'Specify the block number to read
60 TBL%(3)=0                                  :'Specify the device number to read
70 TBL%(4)=10                                 :'Specify the number of points to read
80 PCRD TBL%( ),A%( )                         :'Execute read
90 FOR I=0 TO 9                               :'Display the data that was read out
100  PRINT A%(I)
110 NEXT I
120 END
```

Procedure Code 5 **Monitor an extension file register that was recdorded for monitoring with the PCWT command.**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

- An extension file register that was recorded for monitoring with the **PCWT** command can be monitored, and the results can be read out. If data is read out before the register is recorded for monitoring, an error occurs.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 5 is specified.

PROGRAMMING EXAMPLE

```
10 'Monitor an extension file register that was recorded for monitoring with the PCWT
comand
20 DIM TBL1%(2),A%(3),TBL2%(1),B%(1)          :'Define arrays
30 TBL1%(0)=255                               :'Set the communication station number to
                                               the host
40 TBL1%(1)=5                                 :'Specify "record extension file register
                                               for monitoring"
50 TBL1%(2)=2                                 :'Specify the number of points to monitor
60 A%(0)=1:A%(1)=0                            :'Specify block number 1, device number 0
70 A%(2)=1:A%(3)=1                            :'Specify block number 1, device number 1
80 PCWT TBL1%( ),A%( )                        :'Record for monitoring
90 TBL2%(0)=255                               :'Set the communication station number to
                                               the host
100 TBL2%(1)=5                                :'Specify "monitor extension file register"
110 PCRD TBL2%( ),B%( )                       :'Execute monitoring
120 PRINT "B%(0)=";B%(0)                      :'Display results
130 PRINT "B%(1)=";B%(1)
140 END
```

Procedure Code 7 **Read data from specified consecutive addresses within the extension file register**

for A2A/A3A CPU

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2) [ ]%(3)	device number	Specifies consecutive device numbers within the extension file register
[ ]%(4)	number of points	Specifies the number of points of data to read

- Data can be read from consecutive addresses within the extension file register without separating by block numbers.

- The following data is specified with the **<control table>**:

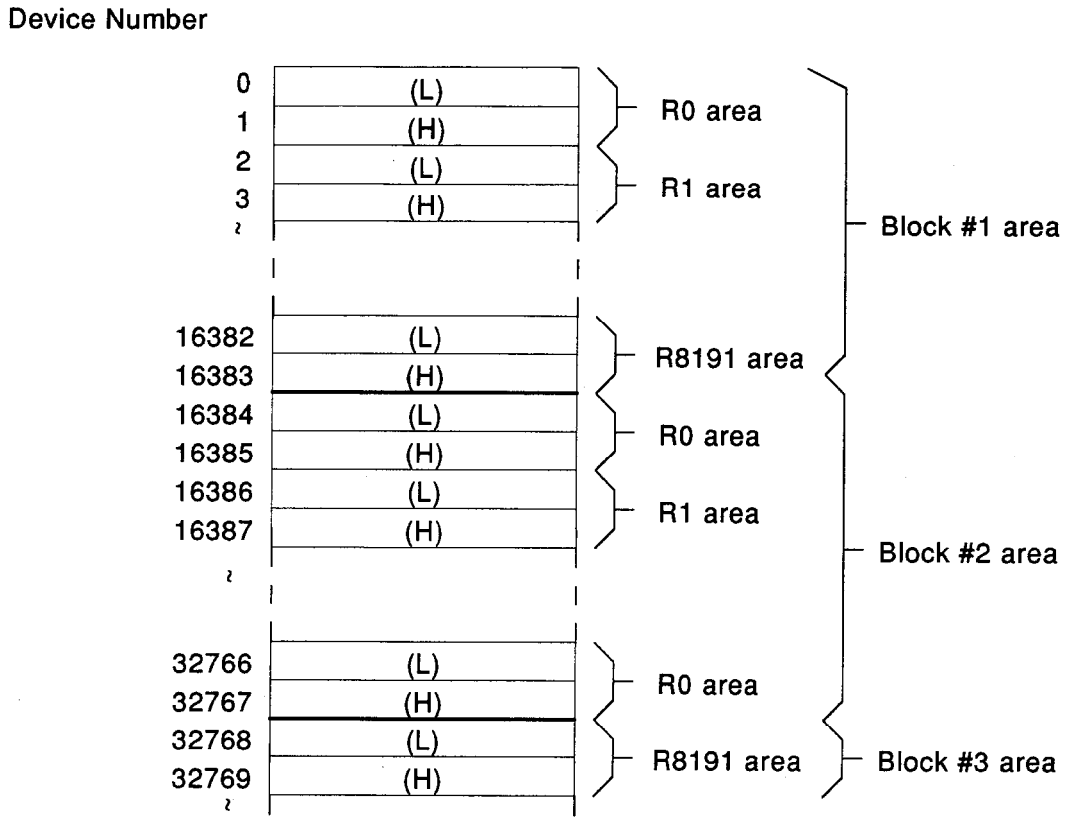
[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

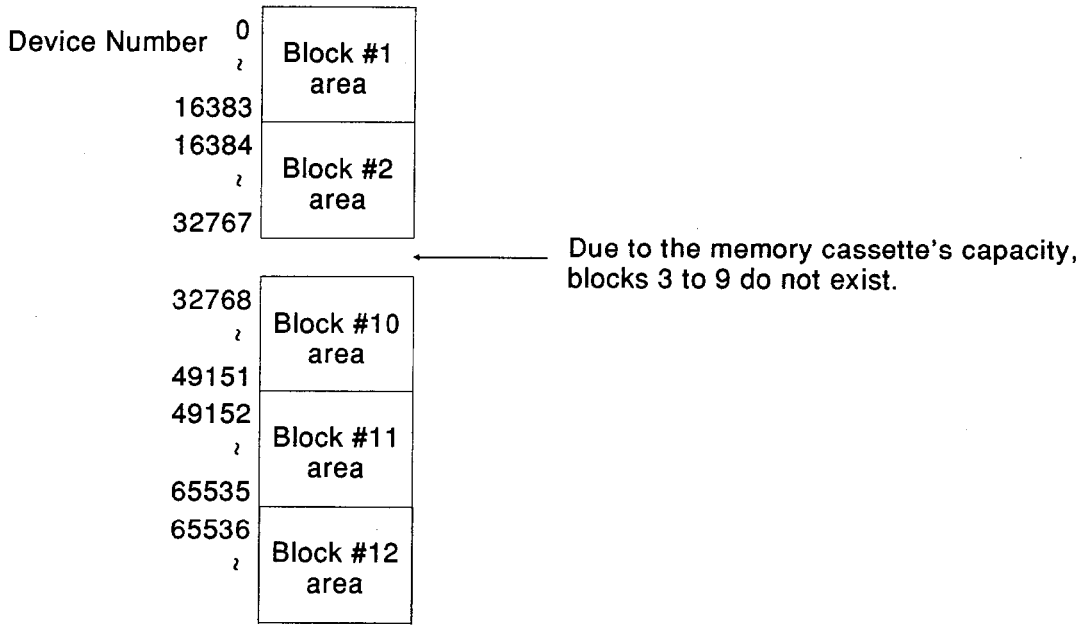
[ ]%(1) Procedure code
- 7 is specified.

[ ]%(2),[ ]%(3) Specifies the device numbers of the extension file register as consecutive addresses.

- The consecutive address device numbers are automatically assigned starting from the low byte of the smallest device of each block number between 1 and 28.



- Device numbers are not assigned to blocks that do not exist on the memory cassette. Therefore, device numbers are assigned automatically, skipping any block numbers that do not exist on the memory cassette.



For information about the relationship between the memory cassette and block numbers, refer to the ACPU Programming Manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions), and the SW[ ]GHP-UTLP-FN1 Operating Manual.

- The value of [ ]%(2) and [ ]%(3) is set as follows:

  D! ..... The consecutive address device number is set.

  H!
  L!
  H$
  L$ — **Used as the work area.**

  ≀

```
100 H!=INT (D!/65536!)
110 L!=D!-H!*65536!
120 H$=RIGHT$("0000"+HEX$(H!))
130 L$=RIGHT$("0000"+HEX$(L!))
140 [ ]% (2)=VAL("&H"+L$)
150 [ ]% (3)=VAL("&H"+H$)
```

  ≀

- [ ]%(4) Specifies how many points of the device are to be read (including the device specified by [ ]%(2) and [ ]%(3)). The range of acceptable values is 1 to 64.

PROGRAMMING EXAMPLE

```
10 'Read the extension file register using consecutive addressing and read
20 DIM TBL%(4),A%(9)                          :'Define arrays
30 YNL%(0)=255                                :'Set the communication station number
                                               to the host
40 TBL%(1)=7                                  :'Select "read the extension file register
                                               by consecutive addressing"
50 D!=0                                       :'Specify the extension file register
                                               consecutive address to read
60 H!=INT(D!/65536!)
70 L!=D!-H!*65536!
80 H$=RIGHT$("0000"+HEX$(H!),4)               :'Define the upper byte of the consecutive
                                               address
90 L$=RIGHT$("0000"+HEX$(L!),4)               :'Define the lower byte of the consecutive
                                               address
100 TBL%(2)=VAL("&H"+L$)                      :'Store the upper byte to the control table
110 TBL%(3)=VAL("&H"+H$)                      :'Store the lower byte to the control table
120 TBL%(4)=10                                :'Read ten points
130 PCRD TBL%( ),A%( )                        :'Display the results of the read
140 FOR I=0 TO 9
150  PRINT A%(I)
160 NEXT I
170 END
```

Procedure Code 8 **Read a sequece program**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	main/sub	Specifies main program or subprogram
[ ]%(3)	head step	Specifies the head step of the program to read
[ ]%(4)	number of steps	Specifies the number of steps to read

- The sequence program inside the PC CPU is read.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.

- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code

- 8 is specified.

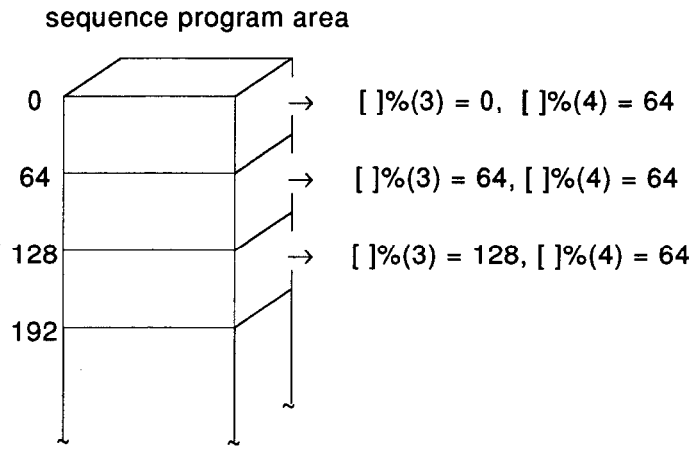
[ ]%(2) To specify main program ... 1

To specify a subprogram ... 2
For CPUs that cannot use subprograms, 1 must be specified.

[ ]%(3) Specifies the head step count of the program area of the program to be read. The capacity of the sequence program is set by a parameter as a step count, so the range of values that can be specified for the head step count is as follows:

0 to 511 × (the sequence program capacity set with the parameter)

[ ]%(4) Specifies the number of steps to read.
64 steps of a sequence program can be read at one time. Therefore, the entire program can be read by specifying the head step count and the number of steps to read so that the program is read 64 steps at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Read the sequence program from the PC CPU and store it on a memory card.
20 OPEN "0:PROG8.DAT" FOR OUTPUT AS #1        :'Open a sequential file
30 DIM TBL%(4),A%(63)                         :'Define arrays
40 TBL%(0)=255                                :'Set the communication station number to
                                               the host
50 TBL%(1)=8                                  :'Select "read sequence program"
60 TBL%(2)=1                                  :'Select "main program"
70 TBL%(4)=64                                 :'Specify the number of steps to read
80 FOR J=0 TO 2*511 STEP 64
90 TBL%(3)=J                                  :'Specify the head step of the program
100 PCRD TBL%( ),A%( )                        :'Execute read
110 FOR I=0 TO 63                             :'Loop I from 0 to 63
120 PRINT #1,A%(I)                            :'Write to sequential file
130 NEXT I
140 NEXT J
150 CLOSE #1                                  :'Close the file
160 END
```

Procedure Code 9 **Read a microcomputer program**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	main/sub	Specifies main program or subprogram
[ ]%(3)	head address	Specifies the head address of the program to read
[ ]%(4)	number of bytes	Specifies the number of bytes to read

- The microcomputer program inside the PC CPU is read.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote station when the AD51H is attached to a I/O master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 9 is specified.

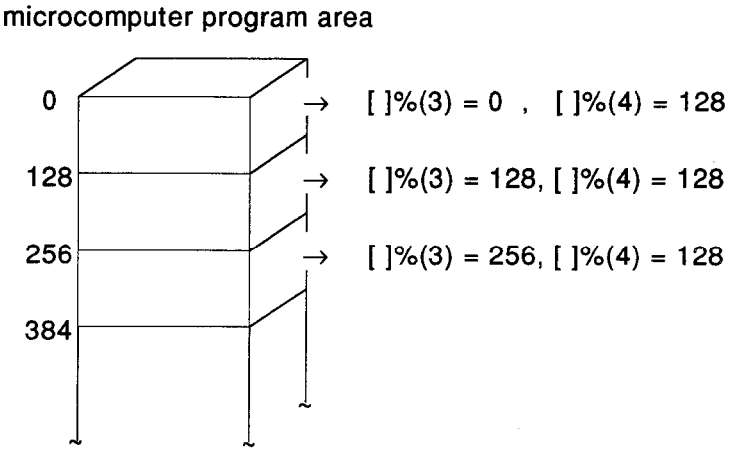
[ ]%(2) To specify main program ... 1
To specify a subprogram ... 2
For CPUs that cannot use subprograms, 1 must be specified.

[ ]%(3) Specifies the head address of the program area of the program to be read.
The capacity of the microcomputer program is set by a parameter in byte units, so the range of values that can be specified for the head address is as follows:

0 to 511 × (the microcomputer program capacity set with the parameter)

[ ]%(4) Specifies the number of bytes to read.

128 bytes of a microcomputer program can be read at one time. Therefore, the entire program can be read by specifying the head address and the number of bytes to read so that the program is read 128 bytes at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Read the microcomputer program from the PC CPU and store it on a memory card.
15 OPEN "0:PROG9.DAT" FOR OUTPUT AS #1        :'Open a sequential file
20 DIM TBL%(4),A%(128)                        :'Define arrays
30 TBL%(0)=225                                :'Set the communication station number
                                               to the host
40 TBL%(1)=9                                  :'Select "read microcomputer program"
50 TBL%(2)=1                                  :'Select "main program"
51 TBL%(4)=128                                :'Specify the number of bytes to read
55 FOR J=0 TO 2*511 STEP 128
60 TBL%(3)=J                                  :'Specify the head address of the program
80 PCRD TBL%( ),A%( )                         :'Execute read
100 FOR I=0 TO 128                            :'Loop I from 0 to 128
110  PRINT #1,A%(I)                           :'Write to the file
120 NEXT I
125 NEXT J
130 CLOSE #1                                  :'Close the file
140 END
```

Procedure Code 10 **Read comment data**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	head address	Specifies the head address of the comment data to read
[ ]%(3)	number of bytes	Specifies the number of bytes to read

- Comment data inside the PC CPU is read.
- The following data is specified with the **<control table>**:

[ ]%(0)

Specifies the station number of the PC CPU from which to read data.

- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H isattached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1)

Procedure code

- 10 is specified.

[ ]%(2)

Specifies the head address of the comment area of the comments to be read.

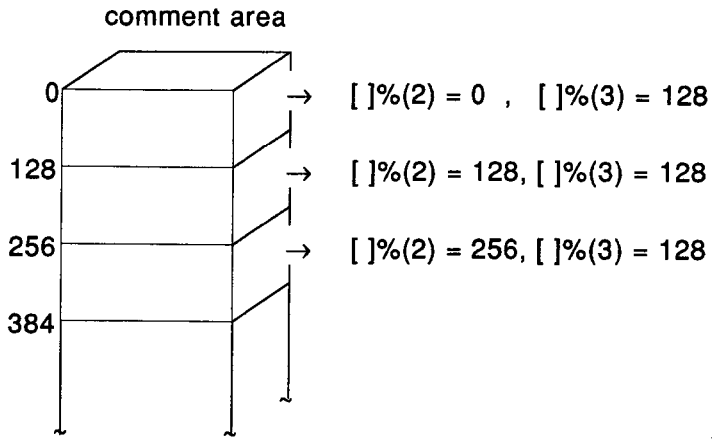
The comment capacity is set by a parameter in byte units, so the range of values that can be specified for the head address is as follows:

0 to 1023 × (the comment capacity set with the parameter)

[ ]%(3)

Specifies the number of bytes to read.

128 bytes of comment data can be read at one time. Therefore, all comment data can be read by specifying the head address and the number of bytes to read so that the data is read 128 bytes at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Read comment data from the PC CPU and store it on a memory card.
20 OPEN "0:PROG10 DAT" FOR OUTPUT AS #1      :'Open a sequential file
30 DIM TBL%(4),A%(128)                       :'Define arrays
40 TBL%(0)=225                               :'Set the communication station number
                                              to the host
50 TBL%(1)=10                                :'Select "read comment data"
60 TBL%(3)=128                               :'Specify the number of bytes to read
70 FOR J=0 TO 3*1023 STEP 128
80 TBL%(2)=J                                 :'Specify the head address of the comment
                                              area
90 PCRD TBL%( ),A%(I)                        :'Execute read
100 FOR I=0 TO 128                           :'Loop I from 0 to 128
110  PRINT #1,A%(I)                          :'Write to the file
120 NEXT I
130 NEXT J
140 CLOSE #1                                 :'Close the file
150 END
```

Procedure Code 11 **Read extension comment data**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	head address	Specifies the head address of the comment data to read
[ ]%(3)	number of bytes	Specifies the number of bytes to read

- Extension comment data inside the A2A(S1) or A3A CPUs is read.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.

- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code

- 11 is specified.

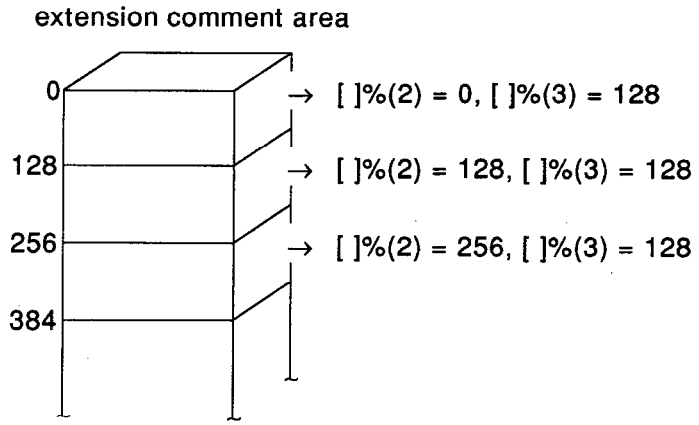
[ ]%(2) Specifies the head address of the extension comment area of the comments to be read.

[ ]%(3) Specifies the number of bytes to read.

The extension comment capacity is set by a parameter in byte units, so the range of values that can be specified for the head address is as follows:

0 to 1023 X (the comment capacity set with the parameter)

128 bytes of extension comment data can be read at one time. Therefore, all extension comment data can be read by specifying the head address and the number of bytes to read so that the data is read 128 bytes at a time, as shown below:



POINT

Extension comments can only be created when a PC/AT running the SW0IX-GPPAE software package is used as the peripheral device. Also, extension comment data cannot be used with the LEDC or similar commands within the sequence program.

PROGRAMMING EXAMPLE

```
10 'Read extension comment data from the A2A(S1), A3A CPU.
20 OPEN "0:PROG11.DAT" FOR OUTPUT AS #1
                                    :'Open a file
30 DIM TBL%(4),A%(128)              :'Define arrays
40 TBL%(0)=255                      :'Set the communication station
                                     number to the host
50 TBL%(1)=11                       :'Select "read extension comment
                                     data"
60 TBL%(3)=128                      :'Specify the number of bytes to
                                     read
70 FOR J=0 TO 3*1023 STEP 128
80 TBL%(2)=J                        :'Specify the head address of the
                                     extension comment area
90 PCRD TBL%( ),A%( )               :'Execute read
100 FOR I=0 TO 128                  :'Loop I from 0 to 128
110 PRINT #1,A%(I)                  :'Write to the file
120 NEXT I
130 NEXT J
140 CLOSE #1                        :'Close the file
150 END
```

Procedure Code 12 **Read the special function module's buffer memory**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	unit number	Specifies the number of the special function module
[ ]%(3)	buffer memory address	Specifies the buffer memory address
[ ]%(4)		
[ ]%(5)	number of bytes	Specifies the number of bytes to read

- The contents of the special function module buffer memory area can be read.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/Ostation 0
  (2) To access a local station or a remote I/Ostation when the AD51H is attached to a master station ... 1 to 64

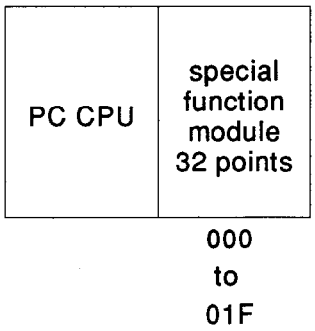
For details see section 4.2.

[ ]%(1) Procedure code
- 12 is specified.

[ ]%(2) Specifies the number of the special function module whose buffer memory contents are to be read.

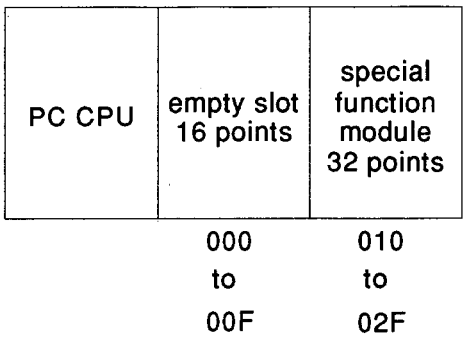
The special function module number (in hexadecimal) is taken from the last address of the I/O addresses occupied by the special function module as seen from the PC CPU side. The actual unit number is represented by truncating the last digit of this 3-digit address.

(a) 1 Slot module (AD61, A68AD, etc.)



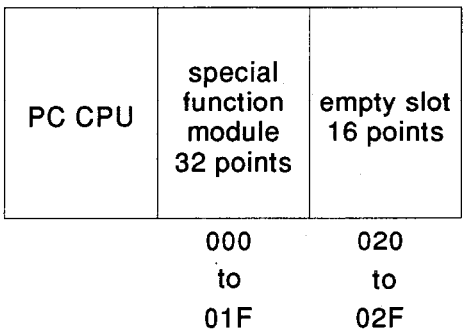
..... The last address is $01F_H$, so the special function module number becomes "01_H"

(b) Modules whose Front Half Slot is Empty (AD72, A84AD, etc.)



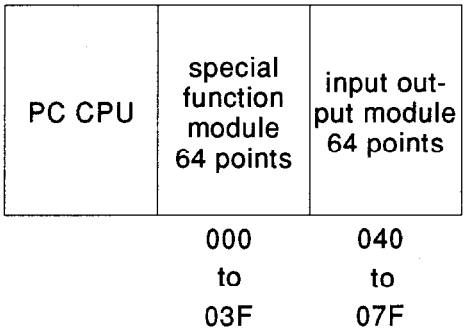
..... The last address used by the special function module is $02F_H$, so the special function module number becomes "02_H"

(c) Modules whose Back Half Slot is Empty (A61LS, etc.)



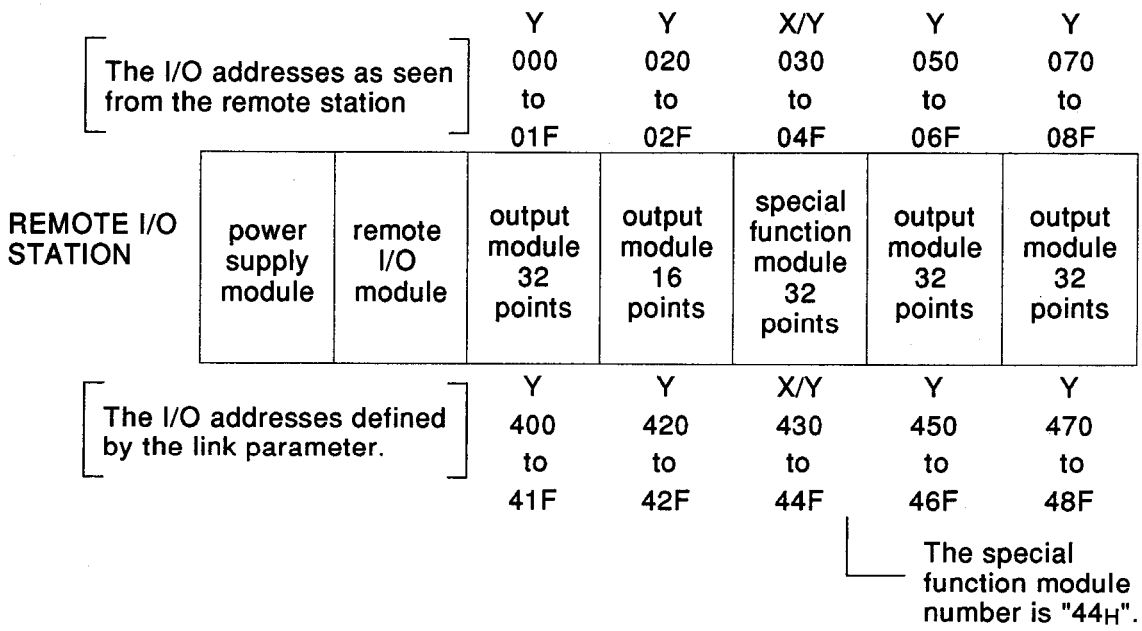
..... The last address used by the special function module is $01F_H$, so the special function module number becomes "01_H"

(d) When the Special Function module is Combined with an Input/Output module
(A81CPU, etc.)



..... The last address used by the special function module is $03F_H$, so the special function module number becomes "03_H"

(e) The special function module number of a MELSECNET remote station is decided by the contents of a link parameter within the MELSECNET master station.



[ ]%(3), [ ]%(4) Specifies the buffer memory address from which to read data.

Each address of the special function module buffer memory is composed of 16 bits (1 word) when read/write operations between the PC CPU and the special function module are performed with the FROM/TO commands. When the AD51H reads from or writes to the special function module's buffer memory, the operation is performed with one address equal to 8 bits (1 byte).

The (hexadecimal) address specified by the AD51H can be calculated from the address used with the FROM/TO commands according to the following equation:

specified address (hexadecimal)	=	{(address used with the FROM/TO commands × 2)} converted to hexadecimal + module head address

Example

To specify address 1 (used by the FROM/TO commands) of the AD61 high speed counter module (channel 1 preset value):

[specified address 82H] = [FROM/TO command address × 2 1Hx2] + [head address 80H]

For information on the head address of each module, see the AD51H(S3) User's Manual.

The values of [ ]%(3) and [ ]%(4) are set as follows:

D! ..... Designated address seen from the AD51H side
H!
L! Used as the work area.
H$
L$

```
100 H!=INT(D!/65536!)
110 L!=D!-H!*65536!
120 H$=RIGHT$("0000"+HEX$(H!), 4)
130 L$=RIGHT$("0000"+HEX$(L!),4)
140 [ ]%(3)=VAL("&H"+L$)
150 [ ]%(4)=VAL("&H"+H$)
```

[ ]%(5) Specifies how many bytes of the special function module's buffer memory are to be read (including the address specified by [ ]%(3) and [ ]%(4)). The range of acceptable values is 1 to 128 bytes.

The Contents of the Data that was Read

Data is stored in the ,**<data storage area>** as shown below.

Example

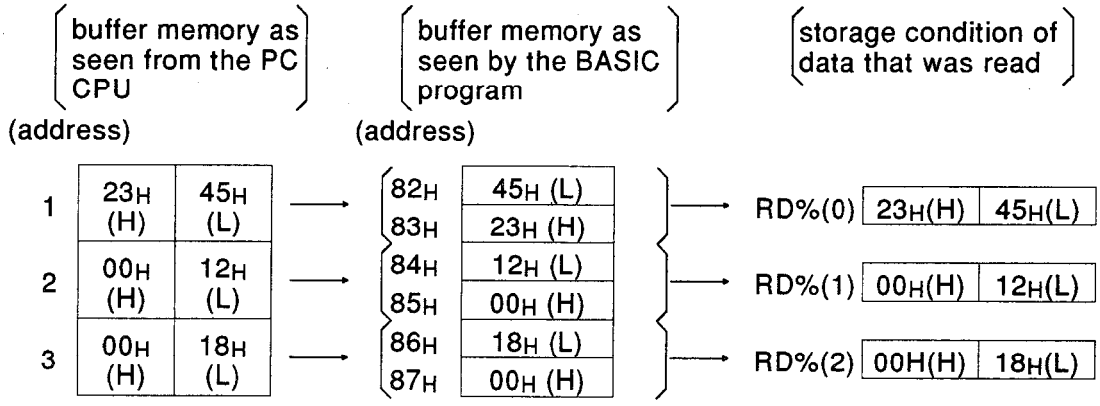
Data is read from buffer memory address regions #1 to #3 of an AD61 module occupying I/O addresses 40 to 5F.

1) If an integer variable or integer array is specified as the input element, data is stored as follows:

PCRD TAB%( ), RD%( )

number of bytes to process ... 6
head address 82H
module number 05H

these parameters are specified

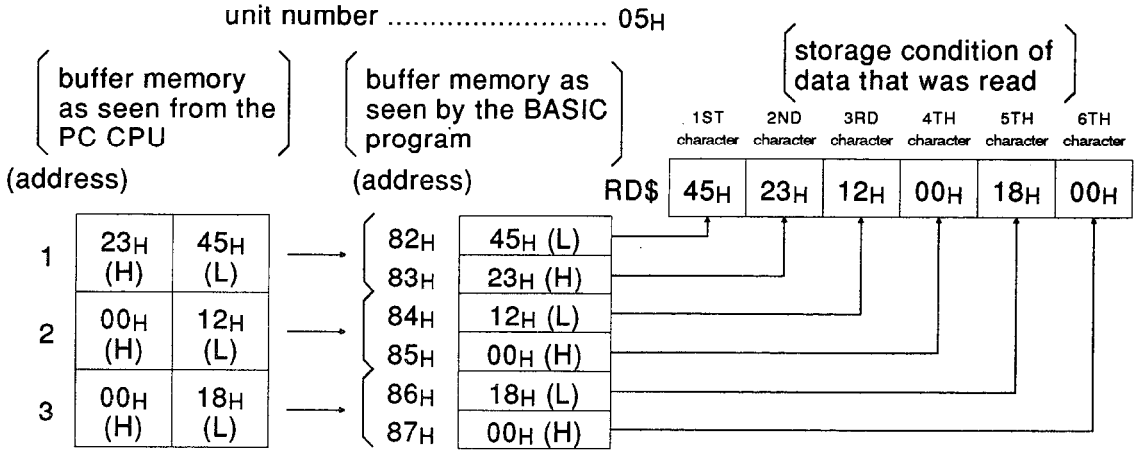


2) If a character string variable or string array is specified as the input element, data is stored as follows:

PCRD TAB%( ), RD%( )

number of bytes to process ... 6
head address 82H
unit number 05H

these parameters are specified



PROGRAMMING EXAMPLE

```
10 'Read the buffer memory of special module A68AD.
20 '(the head address of the A68AD is &H80)
30 DIM TBL%(5),A%(0)                          :'Define arrays
40 TBL%(0)=255                                :'Set the communication station number to
                                               the host
50 TBL%(1)=12                                 :'Select "read buffer memory"
60 TBL%(2)=&H9                                :'Specify the unit number
70 D!=&H94                                    :'Specify the buffer memory address
80 H!=INT(D!/65536!)
90 L!=D!-H!*65536!
100 H$=RIGHT$("0000"+HEX$(H!),4)              :'Define the upper byte of the buffer memory
                                               address
110 L$=RIGHT$("0000"+HEX$(L!),4)              :'Define the lower byte of the buffer memory
                                               address
120 TBL%(3)=VAL("&H"+L$)                      :'Store the upper byte to the control table
130 TBL%(4)=VAL("&H"+H$)                      :'Store the lower byte to the control table
140 TBL%(5)=2                                 :'Specify the number of bytes to read
150 PCRD TBL%( ),A%( )                        :'Execute read
160 PRINT "CH1 DISITAL OUTPUT=";A%(0)         :'Display
170 END
```

Procedure Code 13 **Read the PC CPU type**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

- The type of a PC CPU on the MELSECNET can be read.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 13 is specified.

The PC CPU type is stored in the **<data storage area>** using the following codes:

PC CPU TYPE	CODE (hexadecimal)
A0J2CPU	A0H
A1CPU, A1NCPU	A1H
A2CPU(S1), A2NCPU(S1)	A2H
A3CPU, A3NCPU, A3NCPU-F	A3H
A3HCPU, A3MCPU	A4H
A2ACPU	92H
A2ACPU(S1)	93H
A3ACPU	94H
A0J2HCPU, A1SCPU	98H
A2CCPU	9AH
AJ72(A)P25/R25	ABH

PROGRAMMING EXAMPLE

```
10 'Read the type name of a PC CPU.

20 DIM TBL%(1),A%(0)                          :'Define arrays
30 TBL%(0)=225                                :'Set the communication station number to
                                               the host
40 TBL%(1)=13                                 :'Select "read CPU type name"
50 PCRD TBL%( ),A%( )                         :'Execute read
60 PRINT "The code for the sequencer CPU's type name is &H";HEX$(A%(0))
                                              :'Display type name code
70 END
```

Procedure Code 14 **Read parameter data**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2) [ ]%(3)	head address	Specifies head address of the parameter area to be read
[ ]%(4)	number of bytes	Specifies the number of bytes to read

- Parameter data is read from the PC CPU.
- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU from which to read data.
- To read from the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 14 is specified.

[ ]%(2), [ ]%(3) Specifies the head address of the parameter area to be read.

The values of [ ]%(2) and [ ]%(3) are set as follows:

D! .....The head address of the parameter area.
H!
L!
H$
L$
Used as the work area.

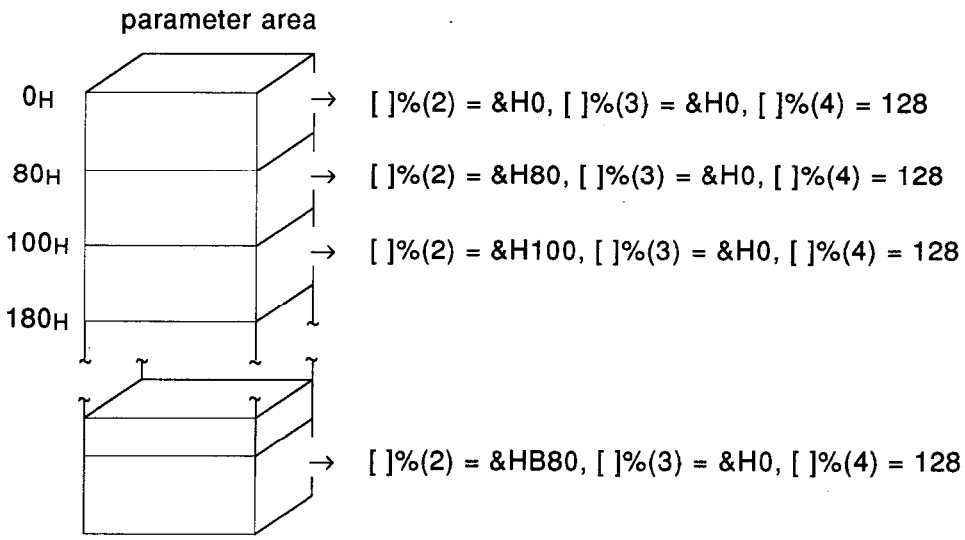
```
100 H!=INT(D!/65536!)
110 L!=D!-H!*65536!
120 H$=RIGHT$("0000"=HEX$(H!),4)
130 L$=RIGHT$("0000"=HEX%(L!),4)
140 [ ]%(2)=VAL("&H"+L$)
150 [ ]%(3)=VAL("&H"+H$)
```

[ ]%(4)

Specifies the number of bytes to read.
The parameter capacity is as follows:

A0J2CPU ..... 16 bytes (addresses 0000_H to $000F_H$)
all other CPUs .... 3K bytes (addresses 0000_H to $0BFF_H$)

128 bytes of parameter data can be read at one time. Therefore, for all CPUs except the A0J2CPU, all of the parameter data can be read by specifying the head address and the number of bytes to read so that the data is read 128 bytes at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Read parameter data from the PC CPU and store it on a memory card.
20 OPEN "0:PROG14.DAT" FOR OUTPUT AS #1        :'Open a sequential file
30 DIM TBL%(4),A%(128)                         :'Define arrays
40 TBL%(0)=225                                 :'Set the communication station number to
                                                the host
50 TBL%(1)=14                                  :'Select "read parameter data"
60 TBL%(4)=128                                 :'Specify the number of bytes to read
70 FOR J=&H0 TO &HBFF STEP 128
80 D!=J                                        :'Specify the head address
90 H!=INT(D!/65536!)
100 L!=D!-H!*65536!
110 H$=RIGHT$("0000"+HEX$(H!),4)               :'Define the upper byte of the head address
120 L$=RIGHT$("0000"+HEX$(L!),4)               :'Define the lower byte of the head address
130 TBL%(2)=VAL("&H"+L$)                       :'Store the upper byte to the control table
140 TBL%(3)=VAL("&H"+H$)                       :'Store the lower byte to the control table
150 PRINT H$,L$
160 PCRD TBL%( ),A%( )                         :'Execute read
170 FOR I=0 TO 128                             :'Loop I from 0 to 128
180  PRINT #1,A%(I)                            :'Write to the file
190 NEXT I
200 NEXT J
210 CLOSE #1                                   :'Close the file
220 END
```

- Writes all types of data to the PC CPU.
- Manipulates PC CPU operation.

PC WriTe

SYNTAX	**PCWTΔ<control table>, <data storage area>**	
	control table	Specifies the type of data to be written to the PC CPU.
	data storage area	Specifies the variable in which to store the data that is to be written to the PC CPU.
EXAMPLE	**PCWT A%( ), B%( )**	The data stored in integer array variable B% is written to the PC CPU according to the contents of integer array A%.

DESCRIPTION

- The **PCWT** command writes all types of data to the PC CPU. It can also be used to manipulate PC CPU operation.
- The details of the procedure are designated with procedure codes in the control table. The procedure details correspond to the procedure codes as follows:

Procedure Code	Procedure Details
1	Write to device memory
2	Record device memory for monitoring
3	Random write to device memory
4	Write to the extension file register
5	Record the extension file register for monitoring
6	Random write to the extension file register
7*1	Write to consecutive addresses of the extension file register
8*2	Write the sequence program
9*2	Write the microcomputer program
10	Write comment data
11	Write extension comment data
12	Write the special function module's buffer memory
14*2	Write parameter data
15*2	Analyze parameter data
16	PC CPU remote stop

17	PC CPU remote run
20	PC CPU interrupt

*1 Procedure code 7 (write to consecutive addresses of the extension file register), is only valid for A2A and A3A CPUs.
*2 Procedure codes 8, 9, 14 and 15 cannot be executed if the corresponding CPU is running.

- The ata storage area specifies where the data to be written to the PC CPU is stored. integer variables, integer arrays, character variables, character arrays, or the internal devices ED and EM can be used. The **<data storage area>**is specified as follows:

integer variable	[ ]%	
character variable	[ ]$	
integer array	[ ]%( )	
character array	[ ]$(n)	Indicated by the special variable.
ED	W@(ED,n)	Indicated by the special variable.
EM	B@(EM,n)	

POINT

Even if the array used as the <data storage area> uses less than ten array elements, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the PCWT command is executed, an error occurs.
(Usually, if fewer than ten elements are required, it is not necessary to define an array with the DIM command.)

- For information on how data to be written is stored, see section 4.2.6.
- Some procedures that are performed when the PCWT command is executed cannot be executed when the PC CPU is running. In the explanations that follow, these procedures will be identified by the following:

  Cannot Execute While CPU is Running

Procedure Code 1 **Write to device memory.**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	units to process	Specifies with what units to process bit devices
[ ]%(3)	device code	Specifies the device code (a device symbol is specified)
[ ]%(4)	device number	Specifies the device number
[ ]%(5)	number of points	Specifies the number of points of data to write

Data can be written to all device memory with the exception of the PC CPU extension file register.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 1 is specified.

[ ]%(2) When writing to a bit device, specifies whether to perform the operation in bit units or word units.
- To specify bit units 1
- To specify word units 2

When writing to a word device, 2 must be specified.

[ ]%(3) Specifies the symbol that has been decided for the device code.
For information on the code numbers, see section 4.2.5.

[ ]%(4) Specifies the device number for the device selected by [ ]%(3).
For the range of device numbers that can be specified, see section 4.2.5.
However, when word unit handling has been selected for a bit device, the device number must be specified as zero or a multiple of 16.

[ ]%(5) Specifies how many points of the device are to be written (including the device specified by [ ]%(3) and [ ]%(4)). The range of acceptable values are as follows:

When bit unit handling is selected ... 1 to 160 points

When word unit handling is selected

- Bit devices ..... 1 to 10 words (16 to 160 points,16 point units)
- Word devices ..... 1 to 64 points

- If the file register (R) is specified as the device to write, the following conditions apply:
  (1) If the block number of the file register has not been changed at all by the PC CPU, the data is written to block number zero of the file register.
  (2) If the block number of the file register has been changed by the PC CPU, the data is written to the block number that was changed.
    - The block number can be changed by the RSET command, which is included in the Microcomputer Program Package of the SW[ ]GHP-UTLPC-FN1 Utilities Package.
    - The block number can also be changed with the RSET command for use only by A2A and A3A CPUs.

POINT

If an A2A or A3A CPU is in use, and the RSET command is used to set the file register block number to a value greater than or equal to 29, procedure code 1 cannot be used. Instead, procedure code 4 (write data to the extended file register) should be used.

When the extension file register is used, in order to be sure that the process is carried out correctly, **procedure code 4** (write data to the extension file register) should be used to write data to the file register.

PROGRAMMING EXAMPLE

```
10   'Write data to the internal relay M0 to M15
20   DIM TBL%(5),A%(0)                        :'Define arrays
30   TBL%(0)=255                              :'Set the communication station number to
                                               the host
40   TBL%(1)=1                                :'Select "write to device memory"
50   TBL%(2)=2                                :'Select word units
60   TBL%(3)=3                                :'Select internal relay (M)
70   TBL%(4)=0                                :'Specify the device number
80   TBL%(5)=1                                :'Specify the number of words to write
90   A%(0)=&H5                                :'Set M1 and M3 ON, all others OFF
100   PCWT TBL%( ),A%( )                      :'Execute write
110   END
```

Procedure Code 2 **Record device memory for monitoring**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	units to process	Specifies with what units to process bit devices
[ ]%(3)	number of points	Specifies the number of points to monitor

PC CPU device memory with the exception of the extension file register is recorded for monitoring. Once device memory is recorded for monitoring, any random device can be easily monitored with the **PCRD** command.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.

- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  - (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  - (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code

- 2 is specified.

[ ]%(2) When monitoring a bit device, specifies whether to perform the operation in bit units or word units.

- To specify bit units 1
- To specify word units 2

When monitoring a word device, 2 must be specified.

[ ]%(3) Specifies how many points of the device are to be monitored. The range of acceptable values are as follows:

When bit unit handling is selected ... 1 to 40 points

When word unit handling is selected

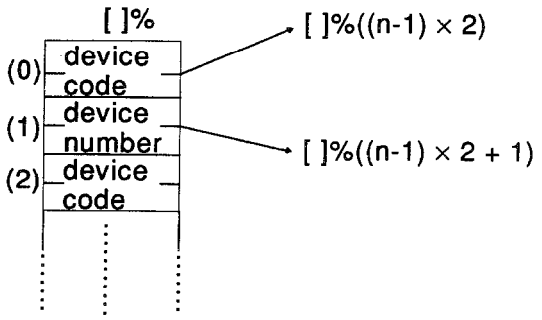
- Bit devices ..... 1 to 20 words (16 to 320 points,16 point units)
- Word devices ..... 1 to 20 words

POINT

For CPUs other than A3HCPU, A2ACPU(S1) and A3ACPU, one point of device X (input) is specified as two points to process.
If the specified device includes X,
((specified number of points of X × 2) +
specified number of points of other devices) ≤ 40 points
If only X is specified, the number of points that can be transferred at one time is half of its normal value.

The devices are set for the specified number of points to monitor with an integer array used as the **<data storage area>** as shown below:

To specify n device points:



Specifies the symbol that has been decided for the device code. For information on the code numbers, see section 4.2.5.

Specifies the device number for the device selected above. For the range of device numbers that can be specified, see section 4.2.5. However, when word unit handling has been selected for a bit device, the device number must be specified as zero or a multiple of 16.

For example, if two points are to be monitored:

specify the first point	[ ]%((1-1) × 2)	→[ ]%(0)	device code
	[ ]%((1-1) × 2 + 1)	→[ ]%(1)	device number
specify the second point	[ ]%((2-1) × 2)	→[ ]%(2)	device code
	[ ]%((2-1) × 2 + 1)	→[ ]%(3)	device number

- If the file register (R) is specified as the device to monitor, when the device is monitored (with the **PCRD** command), the following conditions apply:
  (1) If the block number of the file register has not been changed at all by the PC CPU, the data in block number zero of the file register is monitored.
  (2) If the block number of the file register has been changed by the PC CPU, the data in the block number that was changed is monitored.
    - The block number can be changed by the RSET command, which is included in the Microcomputer Program Package of the SW[ ]GHP-UTLPC-FN1 Utilities Package.
    - The block number can also be changed with the RSET command for use only by A2A and A3A CPUs.

POINT

If an A2A or A3A CPU is in use, and the RSET command is used to set the file register block number to a value greater than or equal to 29, procedure code 2 cannot be used. Instead, procedure code 5 (record the extension file register for monitoring) should be used.

When the extension file register is used, in order to be sure that the process is carried out correctly, procedure code 5 (record the extension file register for monitoring) should be used to record the file register for monitoring.

PROGRAMMING EXAMPLE

```
1    'Record for monitoring
10   DIM A%(9),TBL1%(3)                    :'Define arrays
20   A%(0)=1:A%(1)=0                       :'Monitor X000-X015
30   A%(2)=2:A%(3)=16                      :'Monitor Y016-Y031
40   A%(4)=3:A%(5)=96                      :'Monitor M096-M113
50   A%(6)=7:A%(7)=0                       :'Monitor T000 (contact)
60   A%(8)=18:A%(9)=48                     :'Monitor the contents of the settings in
                                            D048
70   TBL1%(0)=255                          :'Set the communication station number to
                                            the host
80   TBL1%(1)=2                            :'Specify "record a device for monitoring"
90   TBL1%(2)=2                            :'Select word units
100   TBL1%(3)=5                           :'Specify the number of device points to
                                            monitor
110   PCWT TBL1%( ),A%( )                  :'Record for monitoring
120   END
```

Procedure Code 3 **Random write to device memory**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	units to process	Specifies with what units to process the bit device
[ ]%(3)	number of points	Specifies the number of points to write

PC CPU device memory with the exception of the extension file register can be randomly written.

- The following data is specified with the **<control table>**:

[ ]%(0)

Specifies the station number of the PC CPU to be accessed.

- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  - (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  - (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1)

Procedure code

- 3 is specified.

[ ]%(2)

When writing to a bit device, specifies whether to perform the operation in bit units or word units.

- To specify bit units 1
- To specify word units ... 2

When writing to a word device, 2 must be specified.

[ ]%(3)

Specifies how many points of the device are to be written. The range of acceptable values are as follows:

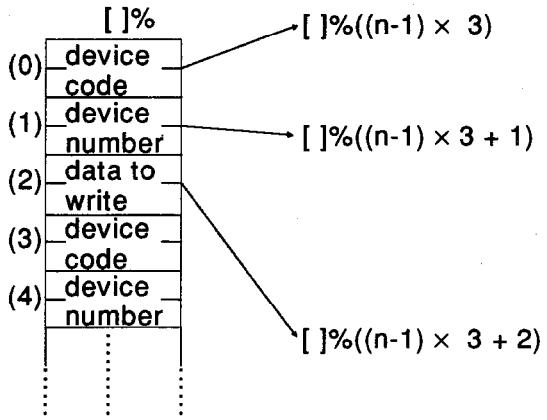
When bit unit handling is selected ... 1 to 20 points

When word unit handling is selected

- Bit devices ..... 1 to 10 words (16 to 160 points,16 point units)
- Word devices ..... 1 to 10 points

The devices and the data to write are set for the specified number of points to write with an integer array used as the **<data storage area>** as shown below:

- To specify n device points:



Specifies the symbol that has been decided for the device code. For information on the code numbers, see section 4.2.5.

Specifies the device number for the device selected above. For the range of device numbers that can be specified, see section .4.2.5 However, when word unit handling has been selected for a bit device, the device number must be specified as zero or a multiple of 16.

Specifies the data to be written to the device. For information on how to specify data, see section 4.2.6.

For example, if two points are to be written:

specify the first point	[ ]%((1-1) × 3)	[ ]%(0)	device code
	[ ]%((1-1) × 3 + 1)	[ ]%(1)	device number
	[ ]%((1-1) × 3 + 2)	[ ]%(2)	data to write
specify the second point	[ ]%((2-1) × 3)	[ ]%(3)	device code
	[ ]%((2-1) × 3 + 1)	[ ]%(4)	device number
	[ ]%((2-1) × 3 + 2)	[ ]%(5)	data to write

- If the file register (R) is specified as the device to write, the following conditions apply:
  (1) If the block number of the file register has not been changed at all by the PC CPU, the data is written to block number zero of the file register.
  (2) If the block number of the file register has been changed by the PC CPU, the data is written to the block number that was changed.
    - The block number can be changed by the RSET command, which is included in the Microcomputer Program Package of the SW[ ]GHP-UTLPC-FN1 Utilities Package.
    - The block number can also be changed with the RSET command for use only by A2A and A3A CPUs.

POINT

If an A2A or A3A CPU is in use, and the RSET command is used to set the file register block number to a value greater than or equal to 29, procedure code 3 cannot be used. Instead, procedure code 6 (random write to the extension file register) should be used.

When the extension file register is used, in order to be sure that the process is carried out correctly, **procedure code 6** (random write to the extension file register) should be used to random write data to the file register.

PROGRAMMING EXAMPLE

```
10  'Random write to device memory
20  DIM TBL%(3),A%(14)                    :'Define arrays
30  TBL%(0)=255                           :'Set the communication station number to
                                           the host
40  TBL%(1)=3                             :'Specify "random write to device memory"
50  TBL%(2)=1                             :'Select bit units
60  TBL%(3)=5                             :'Specify the number of points to write
70  A%(0)=3:A%(1)=0:A%(2)=1               :'Set M00 ON
80  A%(3)=3:A%(4)=5:A%(5)=1               :'Set M05 ON
90  A%(6)=3:A%(7)=10:A%(8)=1              :'Set M10 ON
100  A%(9)=2:A%(10)=0:A%(11)=1            :'Set Y00 ON
110  A%(12)=2:A%(13)=5:A%(14)=1           :'Set Y05 ON
120  PCWT TBL%( ),A%( )                   :'Execute write
130  END
```

Procedure Code 4 **Write data to the extension file register**

Control table components:

Element Position	Item	Details
[ ]%(0)	Station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	block number	Specifies the extension file register block number
[ ]%(3)	device number	Specifies the extension file register device number
[ ]%(4)	number of points	Specifies the number of points to write

Data can be written to the extension file register (all block numbers greater than or equal to 1 within the file register).

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ...1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 4 is specified.

[ ]%(2) Specifies the extension file register block number.
- If zero is specified for the block number, data is written to the PC CPU file register. (This is the same as if device R[ ] is specified for "write device memory.")
- If the value specified for the block number is one or greater, data is written to the extension file register.
- The largest value that can be specified for the block number depends on the memory cassette installed in the PC CPU. For details, see the ACPU Programming Manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions volume), and the SW[ ] GHP-UTLP-FN1 Operating Manual.

[ ] %(3) Specifies the device number of the file register (R). The largest value that can be specified is defined by a PC CPU parameter.

[ ] %(4) Specifies how many points of the device specified in [ ]%(3) are to be written. The range of acceptable values is 1 to 64.

PROGRAMMING EXAMPLE

```
10 'Write data to the extension file register
20 DIM TBL%(4)                          :'Define arrays
30 TBL%(0)=255                          :'Set the communication station number
                                         to the host
40 TBL%(1)=4                            :'Select "write to the extension file register"
50 TBL%(2)=1                            :'Specify the extension file register block
                                         number to write
60 TBL%(3)=0                            :'Specify the file register device number
                                         to write
70 TBL%(4)=1                            :'Specify the number of points to write
80 A%(0)=1234                           :'Specify the data to be written
90 PCWT TBL%( ),A%( )                   :'Execute write
100 END
```

Procedure Code 5 **Record the extension file register for monitoring**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	number of points	Specifies the number of points to monitor

The PC CPU extension file register is recorded for monitoring. Once it is recorded for monitoring, the extension file register can be randomly monitored easily with the **PCRD** command.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.

- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)

  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0

  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

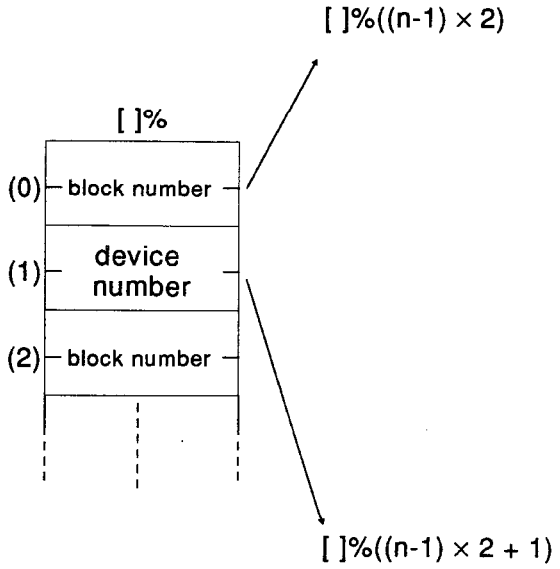
[ ]%(1) Procedure code

- 5 is specified.

[ ]%(2) Specifies how many points of the extension file register are to be monitored. The range of acceptable values is 1 to 20 points.

The extension file register is set for the specified number of points to monitor with an integer array used as the **<data storage area>** as shown below:

To specify n device points:



Specifies the extension file register block number.

- If zero is specified for the block number, the PC CPU file register is recorded for monitoring. (This is the same as if device R[ ] is specified for "record device memory for monitoring.")
- If the value specified for the block number is one or greater, the extension file register is recorded for monitoring.
- The largest value that can be specified for the block number depends on the memory cassette installed in the PC CPU. For details, see the ACPU Programming manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions), and the SW [ ]GHP-UTLP-FN1Operating Manual.

Specifies the device number of the file register (R). The largest value that can be specified is defined by a PC CPU parameter.

For example, if two points are to be monitored:

specify the first point	[ ]%((1-1) × 2)	→[ ]%(0)	block number
	[ ]%((1-1) × 2 + 1)	→[ ]%(1)	device number
specify the second point	[ ]%((2-1) × 2)	→[ ]%(2)	block number
	[ ]%((2-1) × 2 + 1)	→[ ]%(3)	device number

PROGRAMMING EXAMPLE

```
10 'Monitor an extension file register that was recorded for monitoring with the PCWT
   command
20 DIM TBL1%(2),A%(3),TBL2%(1),B%(1)          :'Define arrays
30 TBL1%(0)=255                               :'Set the communication station number to
                                               the host
40 TBL1%(1)=5                                 :'Specify "record extended file register for
                                               monitoring"
50 TBL1%(2)=2                                 :'Specify the number of points to monitor
60 A%(0)=1:A%(1)=0                            :'Specify block number 1, device number 0
70 A%(2)=1:A%(3)=1                            :'Specify block number 1, device number 1
80 PCWT TBL1%( ),A%( )                        :'Record for monitoring
90 TBL2%(0)=255                               :'Set the communication station number to
                                               the host
100 TBL2%(1)=5                                :'Specify "monitor extension file register"
110 PCRD TBL2%( ),B%( )                       :'Execute monitoring
120 PRINT "B%(0)=";B%(0)                      :'Display monitor results
130 PRINT "B%(1)=";B%(1)
140 END
```

Procedure Code 6 **Random write the extension file register**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	number of points	Specifies the number of points to write

The PC CPU extension file register can be randomly written.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.

- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

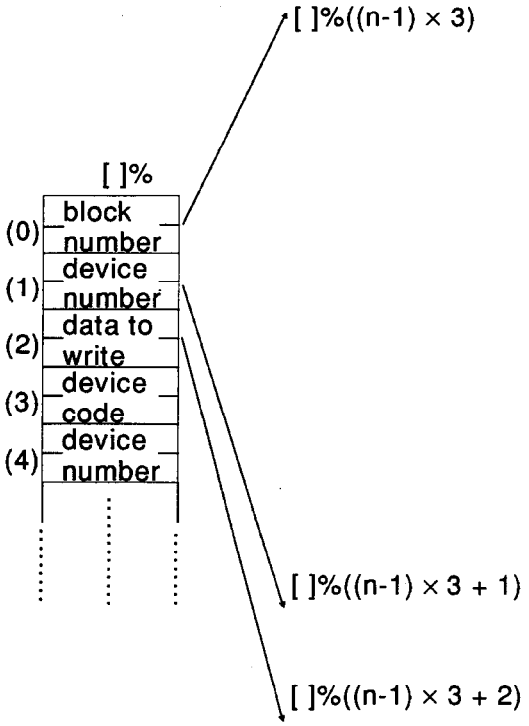
[ ]%(1) Procedure code

- 6 is specified.

[ ]%(2) Specifies how many points of the extension file register are to have data written to them. The range of acceptable values is 1 to 10 points.

The extension file register and the data to write are set for the specified number of points to write with an integer array used as the ata storage area as shown below:

To specify n device points:



Specifies the extension file register block number.

- If zero is specified for the block number,data is written to the PC CPU file (This is the same as if device R[ ] is specified for "write device memory.")
- If the value specified for the block number is one or greater, data is written to the extension file register.
- The largest value that can be specified for the block number depends on the memory-cassette installed in the PC CPU.
  For details, see the ACPU Programming Manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions), and the SW[ ]GHP-UTLP-FN1Operating Manual.

Specifies the device number of the file register (R). The largest value that can be specified is defined by a PC CPU parameter.

Specifies the data to write to the extension file register. For information on how to specify data, see section 4.2.6.

For example, if two points are to be monitored:

specify the first point	[ ]%((1-1) × 3)	→[ ]%(0)	block number
	[ ]%((1-1) × 3 + 1)	→[ ]%(1)	device number
	[ ]%((1-1) × 3 + 2)	→[ ]%(2)	data to write
specify the second point	[ ]%((2-1) × 3)	→[ ]%(3)	block number
	[ ]%((2-1) × 3 + 1)	→[ ]%(4)	device number
	[ ]%((2-1) × 3 + 2)	→[ ]%(5)	data to write

PROGRAMMING EXAMPLE

```
10  'Random write to the extension file register
20  DIM TBL%(2),A%(5)                    :'Define arrays
30  TBL%(0)=255                          :'Set the communication station number to
                                          the host
40  TBL%(1)=6                            :'Specify "random write to the extension file
                                          register"
50  TBL%(2)=2                            :'Specify the number of points to write
60  A%(0)=1:A%(1)=0:A%(2)=1234           :'Specify the block number, device number,
                                          and data to write
70  A%(3)=1:A%(4)=1:A%(5)=5678
80  PCWT TBL%( ),A%( )                   :'Execute write
90  END
```

Procedure Code 7 **Write data to specified consecutive addresses within the extension file register**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2) [ ]%(3)	device number	Specifies consecutive device numbers within the extension file register
[ ]%(4)	number of points	Specifies the number of points to write

Data can be written to consecutive device numbers within the extension file register without separating by block numbers.

- The following data is specified with the **<control table>**:

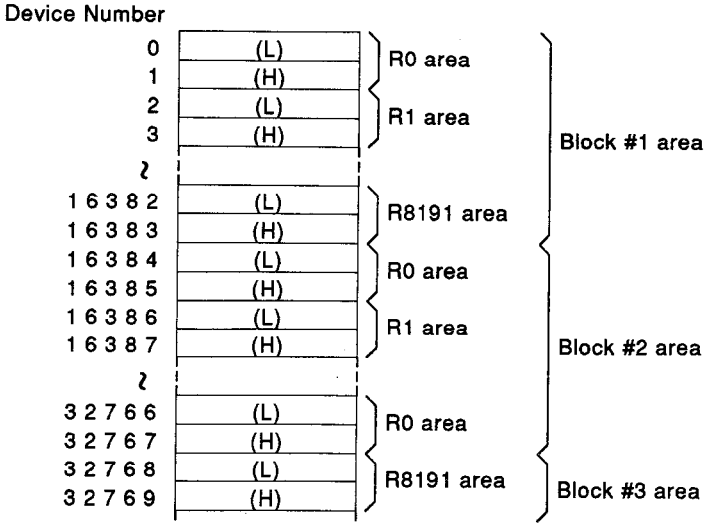
[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

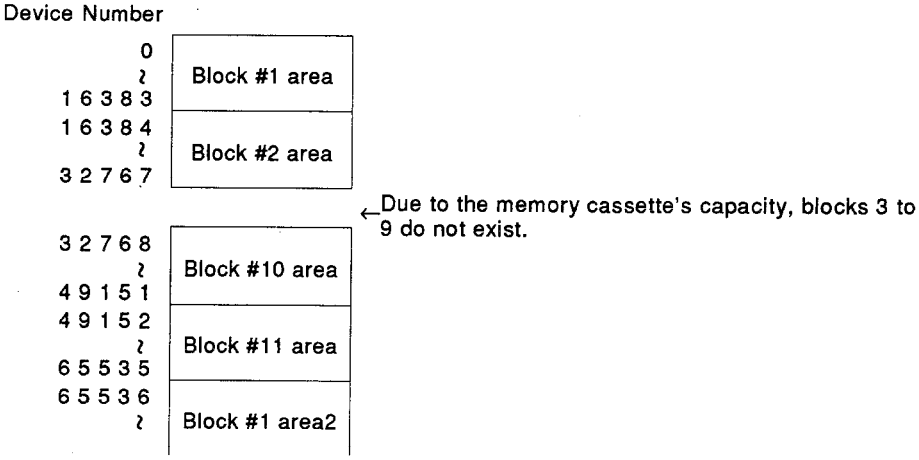
[ ]%(1) Procedure code
- 7 is specified.

[ ]%(2),[ ]%(3) Specifies the device number of the extension file register as consecutive addresses.

The consecutive address device numbers are automatically assigned starting from the low byte of the smallest device of each block number between 1 and 28.



Device numbers are not assigned to blocks that do not exist on the memory cassette. Therefore, device numbers are assigned automatically, skipping any block numbers that do not exist on the memory cassette.



For information about the relationship between the memory cassette and block numbers, refer to the ACPU Programming Manual (Fundamental), the A2A(S1)/A3A Users' Manual (Control Functions), and the SW[ ]GHP-UTLP-FN1 Operating Manual.

The values of [ ]%(2) and [ ]%(3) are set as follows:

D ! ..... The consecutive address device number is set

H !
L !
H $
L $ — Used as the work area.

```
100 H ! = I N T ( D ! / 3 5 5 3 6 ! )
110 L ! = D ! - H ! * 6 5 5 3 6 !
120 H $ = R I G H T $ ( "0 0 0 0" = H E X $ ( H ! ) , 4 )
130 L $ = R I G H T $ ( "0 0 0 0" = H E X $ ( L ! ) , 4 )
140 [ ]%(2) = V A L ( "& H" = L $ )
150 [ ]%(3) = V A L f ( "& H" = H $ )
```

[ ]%(4) Specifies how many points of the device are to be written (including the device specified by [ ]%(2) and [ ]%(3)). The range of acceptable values is 1 to 64.

PROGRAMMING EXAMPLE

```
10  'Specify the extension file register using consecutive addressing and write
20  DIM TBL%(4),A%(3)                          :'Define arrays
30  TBL%(0)=255                                :'Set the communication station number to
                                                the host
40  TBL%(1)=7                                  :'Select "specify extension file register by
                                                consecutive addressing "
50  D!=10                                      :'Specify the consecutive address to read
60  H!=INT(D!/655536!)
70  L!=D!-H!*65536!
80  H$=RIGHT$("0000"+HEX$(H!),4)               :'Define the upper byte of the consecutive
                                                address
90  L$=RIGHT$("0000"+HEX$(L!),4)               :'Define the lower byte of the consecutive
                                                address
100  TBL%(2)=VAL("&H"+L$)                      :'Store the upper byte to the control table
110  TBL%(3)=VAL("&H"+H$)                      :'Store the lower byte to the control table
120  TBL%(4)=3                                 :'Specify the number of points to write
130  A%(0)=1234                                :'Set the data to write
140  A%(1)=5678
150  A%(2)=9101
160  PCWT TBL%( ),A%( )                        :'Execute write
170  END
```

Procedure Code 8 **Write a sequence program** Cannot Execute While CPU is Running

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	main/sub	Specifies main program or subprogram
[ ]%(3)	head address	Specifies the head step of the program to write
[ ]%(4)	number of steps	Specifies the number of steps to write

The sequence program inside the PC CPU is written.

POINT

The sequence program that is written should consist only of data read from the PC CPU using the PCRD command. If other data is used, the PC CPU cannot function properly.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 8 is specified.

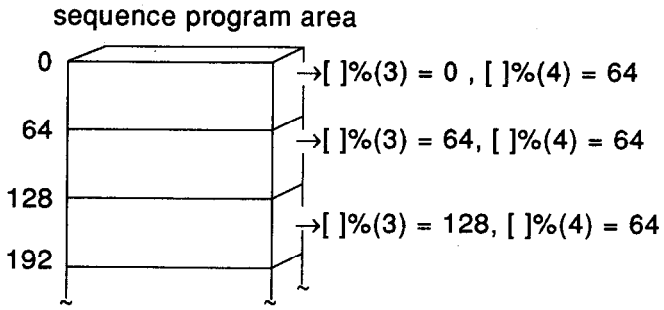
[ ]%(2) To specify main program ... 1
To specify a subprogram ... 2
For CPUs that cannot use subprograms, 1 must be specified.

[ ]%(3) Specifies the head step count of the program area of the program to be written. The capacity of the sequence program is set by a parameter as a step count, so the range of values that can be specified for the head step count is as follows:

0 to 511 × (the sequence program capacity set with the parameter)

[ ]%(4) Specifies the number of steps to write.

64 steps of a sequence program can be written at one time. Therefore, the entire program can be written by specifying the head step count and the number of steps to write so that the program is written 64 steps at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Write a sequence program that is stored on a memory card to the PC CPU.
20 DIM TBL%(4),A%(64)                          :'Define arrays
30 OPEN "0:PROG8.DAT" FOR INPUT AS #1          :'Open a sequential file
40 FOR J=0 TO 2*511 STEP 64
50 FOR I=0 TO 64                               :'Loop I from 0 to 63
60  INPUT #1,A%(I)                             :'Read from sequential file
70 NEXT I
80 TBL%(0)=255                                 :'Set the communication station number
                                                to the host
90 TBL%(1)=8                                   :'Select "write sequence program"
100 TBL%(2)=1                                  :'Select "write to main program area"
110 TBL%(3)=J                                  :'Specify the head step count
120 TBL%(4)=64                                 :'Specify the number of steps to write
130 PCWT TBL%( ),A%( )                         :'Execute write
140 NEXT J
150 CLOSE #1                                   :'Close the file
160 END
```

Procedure Code 9 **Write a microcomputer program** Cannot Execute While CPU is Running

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	main/sub	Specifies main program or subprogram
[ ]%(3)	head address	Specifies the head address of the program to write
[ ]%(4)	number of bytes	Specifies the number of bytes to write

The microcomputer program inside the PC CPU is written.

POINT

The microcomputer program that is written should consist only of data read from the PC CPU using the PCRD command. If other data is used, the PC CPU cannot function properly.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ...1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 9 is specified.

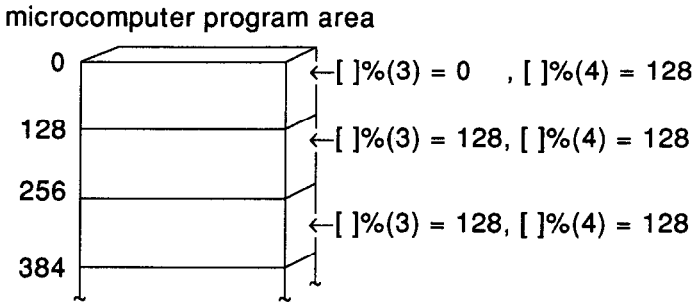
[ ]%(2) To specify main program ... 1
To specify a subprogram ... 2
For CPUs that cannot use subprograms, 1 must be specified.

[ ]%(3) Specifies the head address of the program area of the program to be written.
The capacity of the microcomputer program is set by a parameter in byte units,so the range of values that can be specified for the head address is as follows:

0 to 511 × (the microcomputer program capacity set with the parameter)

[ ]%(4) Specifies the number of bytes to write.

128 bytes of a microcomputer program can be written at one time. Therefore, the entire program can be written by specifying the head address and the number of bytes to write so that the program is written 128 bytes at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Write a microcomputer program that is stored on a memory card to the PC CPU.
20 DIM TBL%(4),A%(128)                    :'Define arrays
30 OPEN "0:PROG9.DAT" FOR INPUT AS #1     :'Open a sequential file
40 FOR I=0 TO 128                         :'Loop I from 0 to 128
50  INPUT #1,A%(I)                        :'Read from the file
60 NEXT I
70 CLOSE #1                               :'Close the file
80 TBL%(0)=255                            :'Set the communication station number
                                           to the host
90 TBL%(1)=9                              :'Select "write microcomputer program"
100 TBL%(2)=1                             :'Select "write to main program area"
110 TBL%(3)=0                             :'Specify the head address
120 TBL%(4)=64                            :'Specify the number of bytes to write
130 PCWT TBL%( ),A%( )                    :'Execute write
140 END
```

Procedure Code 10 **Write comment data**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	head address	Specifies the head address of the comment data to write
[ ]%(3)	number of bytes	Specifies the number of bytes to write

Comment data inside the PC CPU is written.

POINT

The comment data that is written should consist only of data read from the PC CPU using the PCRD command. If other data is used, the PC CPU cannot function properly.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

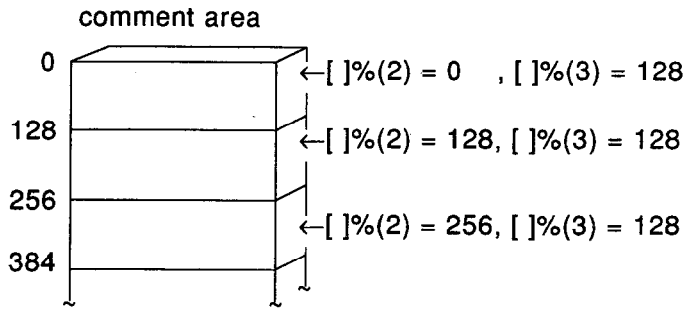
For details see section 4.2.

[ ]%(1) Procedure code
- 10 is specified.

[ ]%(2) Specifies the head address of the comment area of the comments to be written. The comment capacity is set by a parameter in byte units, so the range of values that can be specified for the head address is as follows:

0 - 1023 × (the comment capacity set with the parameter)

[ ]%(3) Specifies the number of bytes to write.
128 bytes of comment data can be written at one time. Therefore, all comment data can be written by specifying the head address and the number of bytes to write so that the data is written 128 bytes at a time, as shown below:



PROGRAMMING EXAMPLE

```
10 'Write comment data that is stored on a memory card to the PC CPU.
20 DIM TBL%(3),A%(128)                    :'Define arrays
30 OPEN "0:PROG10.DAT" FOR INPUT AS #1    :'Open a sequential file
40 FOR J=0 TO 3*1023 STEP 128
50 FOR I=0 TO 128                         :'Loop I from 0 to 128
60  INPUT #1,A%(I)                        :'Read from the file
70 NEXT I
80 TBL%(0)=255                            :'Set the communication station number to the host
90 TBL%(1)=10                             :'Select "write comment data"
100 TBL%(2)=J                             :'Specify the head address
110 TBL%(3)=128                           :'Specify the number of bytes to write
120 PCWT TBL%( ),A%( )                    :'Execute write
130 NEXT J
140 CLOSE #1                              :'Close the file
150 END
```

Procedure Code 11 **Write extension comment data**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	head address	Specifies the head address of the comment data to write
[ ]%(3)	number of bytes	Specifies the number of bytes to write

Extended comment data inside the A2A(S1) or A3A CPUs is written.

POINT

The extension comment data that is written should consist only of data read from the PC CPU using the PCRD command. If other data is used, the PC CPU cannot function properly.

- The following data is specified with the **<controll table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H isconnected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/Ostation ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 11 is specified.

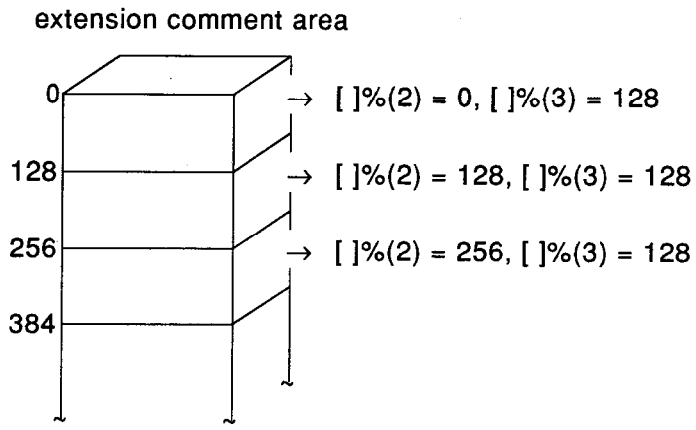
[ ]%(2) Specifies the head address of the extension comment area of the comments to be written.

The extension comment capacity is set by a parameter in byte units, so the range of values that can be specified for the head address is as follows:

0 to 1023 X (the comment capacity set with the parameter)

[ ]%(3) Specifies the number of bytes to write.

128 bytes of extension comment data can be written at one time. Therefore, all extension comment data can be written by specifying the head address and the number of bytes to write so that the data is written 128 bytes at a time, as shown below:



POINT

Extension comments can only be created when a PC/AT running the SW0IX-GPPAE software package is used as the peripheral device. Also, extension comment data cannot be used with the LEDC or similar commands within the sequence program.

PROGRAMMING EXAMPLE

```
10 'Write extension comment data that is stored on a memory card to the
   sequencer CPU.
20 DIM TBL%(3),A%(128)                    :'Define arrays
30 OPEN "0:PROG11.DAT" FOR INPUT AS #1
                                          :'Open a file
40 FOR J=0 TO 3*1023 STEP 128
50 FOR I=0 TO 128                         :'Loop I from 0 to 128
60 INPUT #1,A%(I)                         :'Read from the file
70 NEXT I
80 TBL%(0)=255                            :'Set the communication station
                                           number to the host
90 TBL%(1)=11                             :'Select "write extension comment
                                           data"
100 TBL%(2)=J                             :'Specify the head address
110 TBL%(3)=128                           :'Specify the number of bytes to
                                           write
120 PCWT TBL%( ),A%( )                    :'Execute write
130 NEXT J
140 CLOSE #1                              :'Close the file
150 END
```

Procedure Code 12 **Write to the special function module's buffer memory**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2)	unit number	Specifies the number of the special function module
[ ]%(3) [ ]%(4)	buffer memory address	Specifies the buffer memory address
[ ]%(5)	number of bytes	Specifies the number of bytes to write

The special function module buffer memory area can be written to.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be accessed.
- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

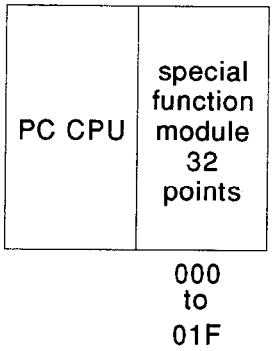
For details see section 4.2.

[ ]%(1) Procedure code
- 12 is specified.

[ ]%(2) Specifies the number of the special function module to whose buffer memory data is to be written.

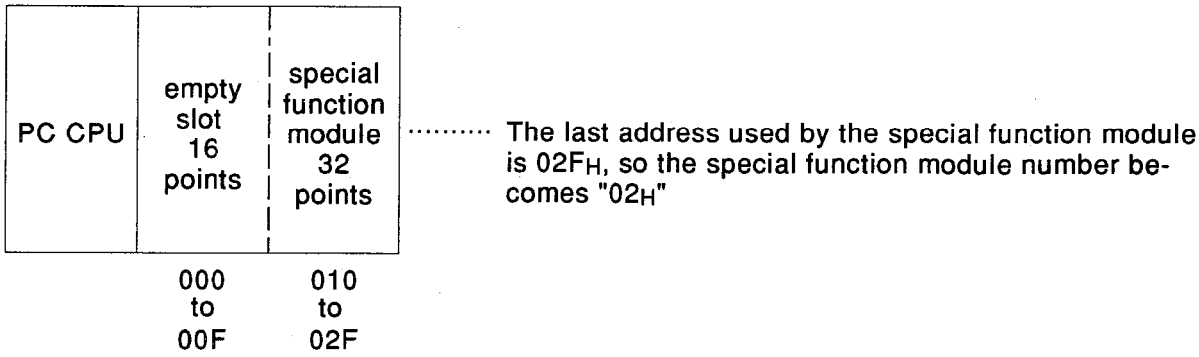
The special function module number (in hexadecimal) is taken from the last address of the I/O addresses occupied by the special function module as seen from the PC CPU side. The actual module number is represented by truncating the last digit of this 3-digit address.

(a) 1 Slot module (AD61, A68AD, etc.)

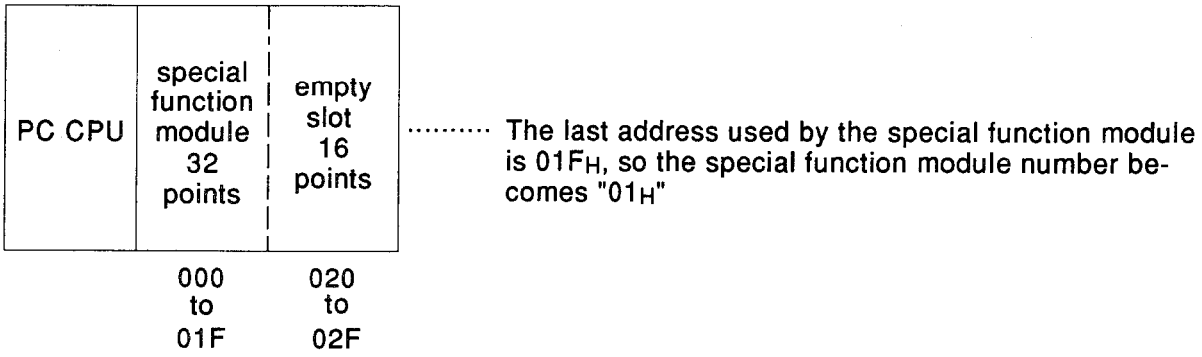


..... The last address is $01F_H$, so the special function module number becomes "01_H"

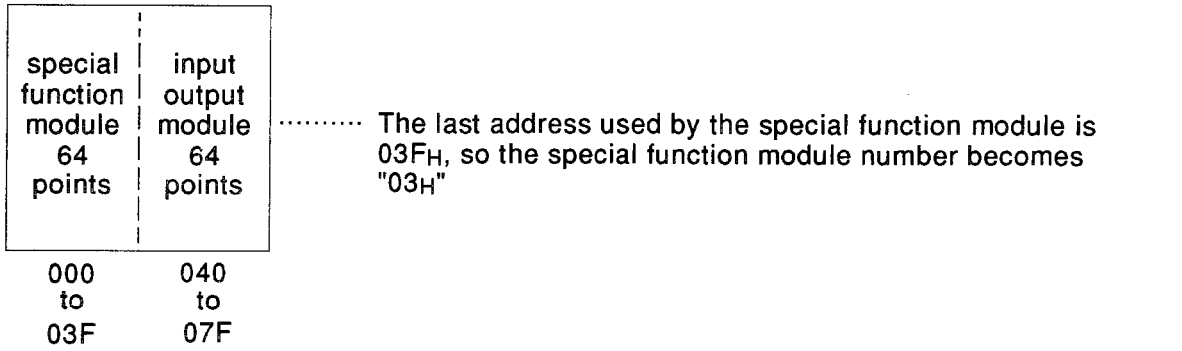
(b) Modules whose front half slot is empty (AD72, A84AD, etc.)



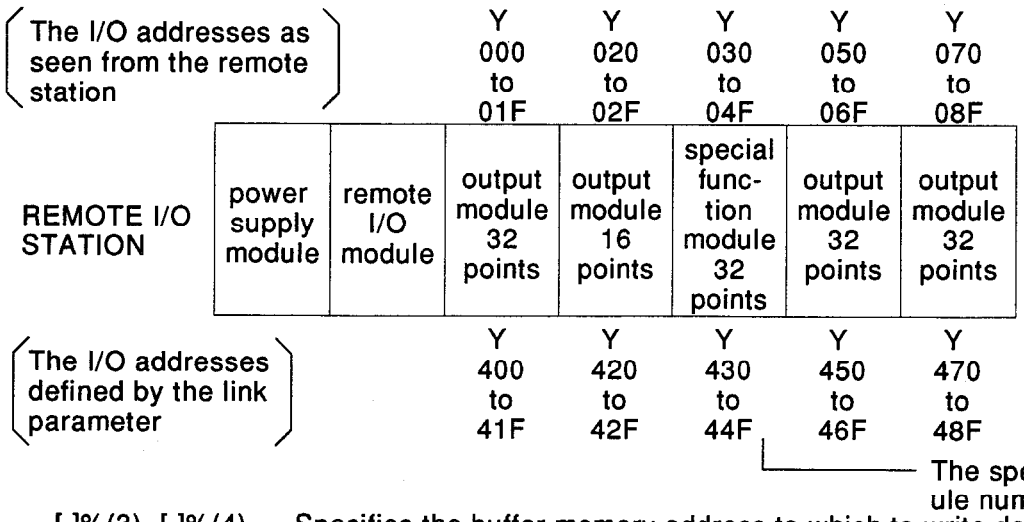
(c) Modules whose back half slot is empty (A61LS, etc.)



(d) When the special function module is combined with an input/output module (A81CPU,etc.)



(e) The special function module number of a MELSECNET remote station is decided by the contents of a link parameter within the MELSECNET master station.



[ ]%(3), [ ]%(4) Specifies the buffer memory address to which to write data.

Each address of the special function module's buffer memory is composed of 16 bits(1 word) when read/write operations between the PC CPU and the special function module are performed with the FROM/TO commands. When the AD51H reads from or writes to the special function module's buffer memory, the operation is performed with one address equal to 8 bits (1 byte).

The (hexadecimal) address specified by the AD51H can be calculated from the address used with the FROM/TO commands according to the following equation:

specified address (hexadecimal) = {(address used with the FROM/TO commands × 2)} converted to hexadecimal + unit head address

Example

To specify address 1 (used by the FROM/TO commands) of the AD61 high speed counter module (channel 1 preset value):

(specified address 82H) = (FROM/TO command address 0×2 1Hx2) + (head address 80H)

- For information on the head addresses of special function modules, see the AD51H/AD51H-S3 User's Manual.

The values of [ ]%(3) and [ ]%(4) are set as follows:

D! ····· Designated address seen from the AD51H side
H!
L!
H$
L$
Used as the work area.

```
100 H!=INT(D!/35536!)
110 L!=D!-H! *65536!
120 H$=RIGHT$("0000"+HEX$(H!),4)
130 L$=RIGHT$("0000"+HEX$(L!),4)
140 [ ]%(3)=VAL("&H"+L$)
150 [ ]%(4)=VAL("&H"+H$)
```

[ ]%(5) Specifies how many bytes of data are to be written to the special function module's buffer memory (starting with the address specified by [ ]%(3) and [ ]%(4). The range of acceptable values is 1 to 128 bytes.

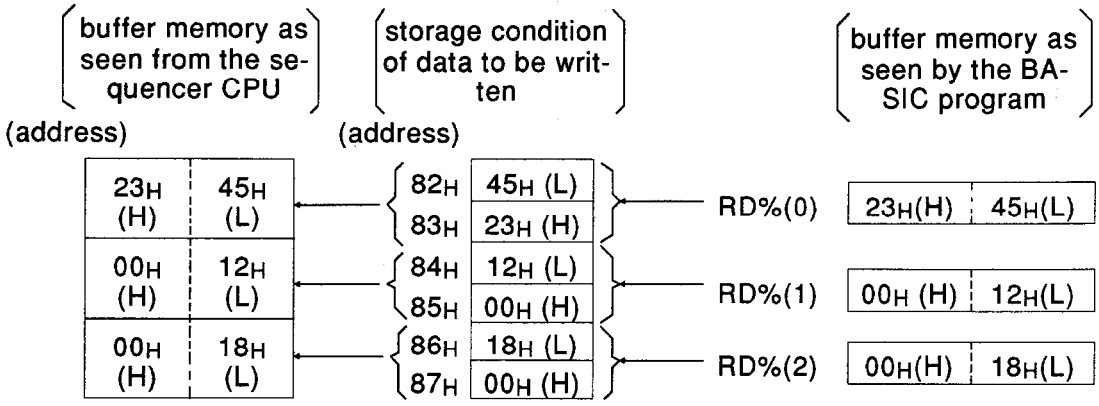
Data is stored in the **<data storage area>** as shown below.

Example
Data is written of buffer memory address regions #1 to #3 of an AD61 module occupying I/O addresses 40 to 5F.

(1) If an integer variable or integer array is specified as the output element, data is stored as follows:

PCWT TAB%( ), RD%( )

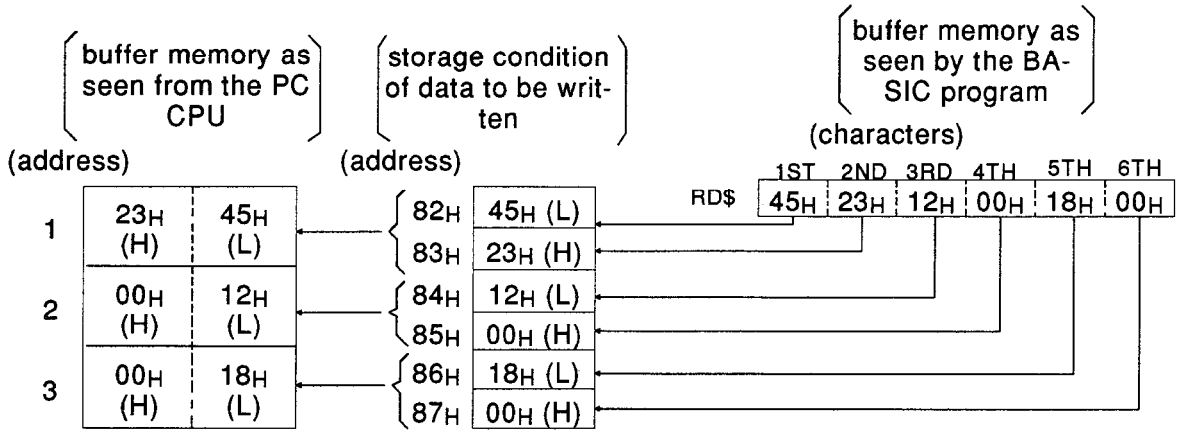
number of bytes to process ... 6
head address 82H
unit number 05H
these parameters are specified



(2) If a character string variable or string array is specified as the output element, data is stored as follows:

PCWT TAB%( ), RD$

number of bytes to process 6
head address 82H
module number 05H
these parameters are specified



PROGRAMMING EXAMPLE

```
10 'Write to the buffer memory of special module A62DA.
20 '(the head address of the A62DA is &H0A)
30 DIM TBL%(5),A%(0)                          :'Define arrays
40 TBL%(0)=255                                :'Set the communication station number
                                               to the host
50 TBL%(1)=12                                 :'Select "write to buffer memory"
60 TBL%(2)=&HA                                :'Specify the module number
70 D!=&HA0                                    :'Specify the buffer memory address
80 H!=INT(D!/65536!)
90 L!=D!-H!*65536!
100 H$=RIGHT$("0000"+HEX$(H!),4)              :'Define the upper byte of the buffer
                                               memory address
110 L$=RIGHT$("0000"+HEX$(L!),4)              :'Define the lower byte of the buffer
                                               memory address
120 TBL%(3)=VAL("&H"+L$)                      :'Store the upper byte to the control table
130 TBL%(4)=VAL("&H"+H$)                      :'Store the lower byte to the control table
140 TBL%(5)=2                                 :'Specify the number of bytes to write
150 A%(0)=&H3E8                               :'Define the value to write
160 PCWT TBL%( ),A%( )                        :'Execute write
170 END
```

Procedure Code 14 **Write parameter data** Cannot Execute While CPU is Running

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code
[ ]%(2) [ ]%(3)	head address	Specifies head address of the parameter area to be written
[ ]%(4)	number of bytes	Specifies the number of bytes to write

Parameter data is written to the PC CPU.

POINT

The parameter data that is written should consist only of data read from the PC CPU using the PCRD command. If other data is used, the PC CPU cannot function properly.

- The following data is specified with the **<control table>:**

[ ]%(0) Specifies the station number of the PC CPU to be accessed.

- To access the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code

- 14 is specified.

[ ]%(2), [ ]%(3) Specifies the head address of the parameter area to be written.

The values of [ ]%(2) and [ ]%(3) are set as follows:

D! ..... The head address of the parameter area.

H!
L!
H$
L$
Used as the work area.

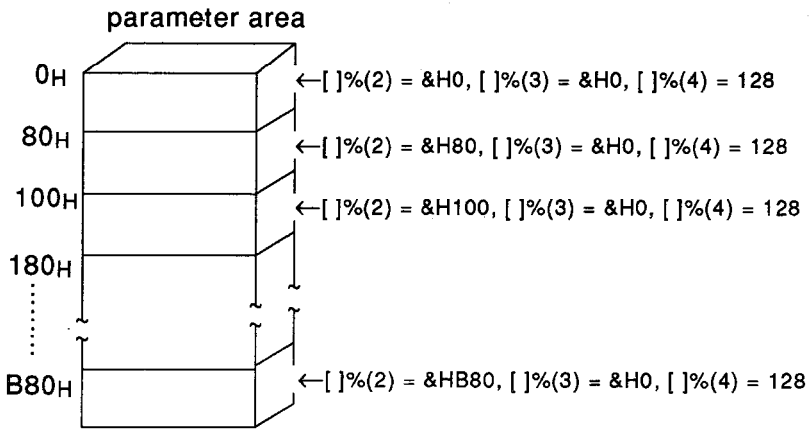
```
100 H!=INT(D!/35536!)
110 L!=D!-H!*65536
120
H$=RIGHT$("0000"+HEX$(L!),4)
130
L$=RIGHT$("0000"+HEX$(L!),4)
140 [ ]%(2)=VAL("&H"+L$)
150 [ ]%(3)=VAL("&H"+H$)
```

[ ]%(4) Specifies the number of bytes to write.
The parameter capacity is as follows:

A0J2CPU ..... 16 bytes (addresses 0000_H to $000F_H$)

all other CPUs ..... 3K bytes (addresses 0000_H to $0BFF_H$)

128 bytes of parameter data can be written at one time. Therefore, for all CPUs except the A0J2CPU, all of the parameter data can be written by specifying the head address and the number of bytes to write so that the data is written 128 bytes at a time, as shown below:



After writing parameter data, it is necessary to perform procedure code 15, "Analyze Parameter Data" →

PROGRAMMING EXAMPLE

```
10 'Write parameter data that is stored on a memory card to the PC CPU.
20 DIM TBL%(4),A%(128)                          :'Define arrays
30 OPEN "0:PROG14.DAT" FOR INPUT AS #1          :'Open a sequential file
40 FOR J=0 TO &HBFF STEP 128
50 FOR I=0 TO 128                               :'Loop I from 0 to 128
60  INPUT #1,A%(I)                              :'Read from the file
70 NEXT I
80 TBL%(0)=255                                  :'Set the communication station number
                                                 to the host
90 TBL%(1)=14                                   :'Select "write parameter data"
100 D!=J                                        :'Specify the head address
110 H!=INT(D!/65536!):'
120 L!=D!-H!*65536!:'
130 H$=RIGHT$("0000"+HEX$(H!),4)                :'Define the upper byte of the head address
140 L$=RIGHT$("0000"+HEX$(L!),4)                :'Define the lower byte of the head address
150 TBL%(2)=VAL("&H"+L$)                        :'Store the upper byte to the control table
160 TBL%(3)=VAL("&H"+H$)                        :'Store the lower byte to the control table
170 TBL%(4)=128                                 :'Specify the number of bytes to write
180 PCWT TBL%( ),A%( )                          :'Execute write
190 NEXT J
200 CLOSE #1                                    :'Close the file
210 TBL%(1)=15                                  :'Specify "analyze parameter data"
220 PCWT TBL%( ),A%( )                          :'Execute analysis
230 END
```

Procedure Code 15 **Analyze parameter data** Cannot Execute While CPU is Running

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

After parameter data is written, it is necessary to tell the PC CPU that parameter data has changed. Therefore, parameter analysis is performed.

After parameter data has been written, the **<control table>** data is changed as follows:

[ ]%(0) No changes are necessary, remains as before.

[ ]%(1) Procedure code
- 15 is specified.

[ ]%(2), [ ]%(3) No changes are necessary, remain as before.

A dummy variable or array is specified for the **<data storage area>**. No matter what kind of variable or array is specified, the process is carried out the same. Also, the values inside the variable or array are not changed during the operation.

POINT

If the PC CPU is not informed that the parameters have been changed, operations will continue with the same parameter contents as before the change.

Procedure Code 16 **PC CPU remote STOP**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

A PC CPU remote STOP can be performed.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to be remote stopped.
- For the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 16 is specified.

A dummy variable or array is specified for the **<data storage area>**. No matter what kind of variable or array is specified, the process is carried out the same. Also, the values inside the variable or array are not changed during the operation.

PROGRAMMING EXAMPLE

```
10 'Perform a PC CPU remote STOP
20 DIM TBL%(1),A%(0)                    :'Define arrays
30 TBL%(0)=255                          :'Set the communication station number to
                                         the host
40 TBL%(1)=16                           :'Specify "PC CPU remote STOP"
50 PCWT TBL%( ),A%( )                   :'Execute
60 END
```

Procedure Code 17 **PC CPU remote RUN**

Control table components:

Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

A PC CPU remote RUN can be performed.

- The following data is specified with the **<control table>**:

[ ]%(0) Specifies the station number of the PC CPU to remote run.
- For the PC CPU that the AD51H is connected to ... 255
- To access PC CPUs via MELSECNET (II)
  (1) To access a master station when the AD51H is attached to a local station or a remote I/O station ... 0
  (2) To access a local station or a remote I/O station when the AD51H is attached to a master station ... 1 to 64

For details see section 4.2.

[ ]%(1) Procedure code
- 17 is specified.

A dummy variable or array is specified for the **<data storage area>**. No matter what kind of variable or array is specified, the process is carried out the same. Also, the values inside the variable or array are not changed during the operation.

PROGRAMMING EXAMPLE

```
10 'Perform a PC CPU remote RUN
20 DIM TBL%(1),A%(0)                    :'Define arrays
30 TBL%(0)=255                          :'Set the communication station number to
                                         the host
40 TBL%(1)=17                           :'Specify "PC CPU remote RUN"
50 PCWT TBL%( ),A%( )                   :'Execute
60 END
```

Procedure Code 20 **Issue a PC CPU interrupt**

Control table components:

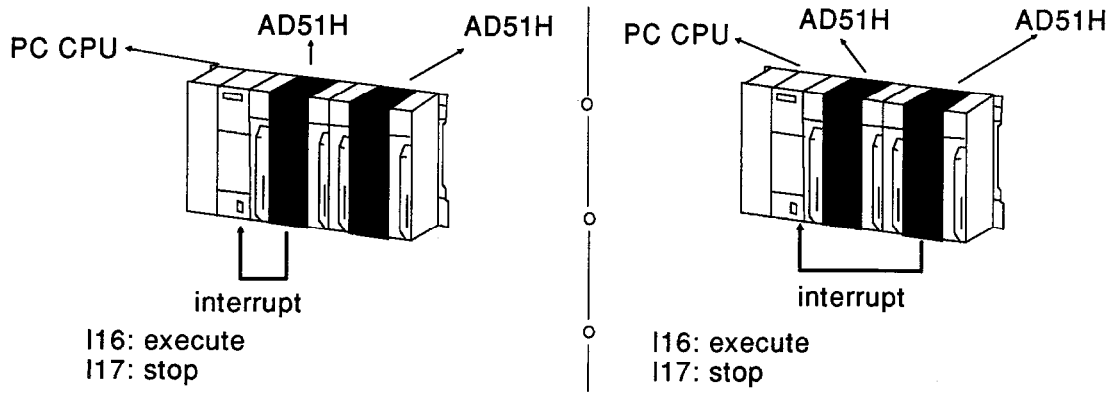
Element Position	Item	Details
[ ]%(0)	station number	Specifies the PC's station number
[ ]%(1)	procedure code	Specifies the procedure code

An interrupt can be issued to the PC CPU to which the AD51H is attached. When an interrupt is issued to the PC CPU by the AD51H, the PC CPU executes the following interrupt programs.

A2A(S1)/A3ACPU Interrupt program at I16 to I21
All other CPUs Interrupt program at I16 • I17

Which program is executed depends on the position in which the AD51H is installed in the PC CPU.

Example
Two AD51H module may be installed in one A3NCPU. If an interrupt is sent from the closer AD51H module, interrupt program I16 is executed, if the interrupt comes from the further AD51H module, interrupt program I17 is executed.



If only one AD51H module is installed, interrupt program I16 is executed.

POINT

If an AD51H and a special function module that can send interrupts to the PC CPU are both installed, the closer module causes interrupt program I16 to be executed.

For information on creating interrupt programs, see the ACPU Programming Manual (Fundamental).

- The following data is specified with the **<control table>**:

[ ]%(0) Normally set to 255.

[ ]%(1) Procedure code
- 20 is specified.

A dummy variable or array is specified for the **<data storage area>**. No matter what kind of variable or array is specified, the process is carried out the same. Also, the values inside the variable or array are not changed during the operation.

PROGRAMMING EXAMPLE

```
10 'Send an interrupt to the PC CPU
20 DIM TBL%(1),A%(0)                    :'Define arrays
30 TBL%(0)=255                          :'Set the communication station number
                                         to the host
40 TBL%(1)=20                           :'Specify "interrupt"
50 PCWT TBL%( ),A%( )                   :'Execute interrupt
60 END
```

- Displays data on the screen.

PRINT

SYNTAX	**PRINTΔ[<data to display>] [;]**

	data to display	Specifies a numeric value, numeric variable, character string, or character variable to display.
EXAMPLE	**PRINT "ABC"**	ABC is displayed.
	PRINT "ABC","DEF"	ABC DEF is displayed. 14 chars. 14 chars.
	PRINT "ABC";"DEF"	ABCDEF is displayed.

DESCRIPTION

- The **PRINT** command displays data on the screen.
- Data is displayed on the console or the terminal designated with the **ZODV** command.
- If **<data to display>** is omitted, one line of whitespace is displayed.
- Character strings must be enclosed in quotes.
- The display position of the **<data to display>** is decided with delimiter symbols.
  In BASIC, each line is divided into 14 character areas.
  If the **<data to display>** is separated by commas, the next **<data to display>** is displayed at the beginning of the next available area.
  If the semicolon is used as the delimiter, the next character string is attached to the end of the previous one.
  When displaying except the console screen, the comma cannot be used to separate **<data to print>**.
  If the comma is used, an error occurs.
  If whitespace is used to separate **<data to display>** it is handled the same as if a semicolon had been used.
- If a comma is entered at the end of the **PRINT** command following the last **<data to display>**, when the next **PRINT** command is executed, data is displayed in the next area. If a semicolon is entered at the end of the PRINT command following the last character string, when the next PRINT command is executed, data is displayed immediately following the previous **PRINT** command's data.

PRINT "ABC",
PRINT "DEF",

```
ABC      DEF
14 chars. 14 chars.
```

PRINT "ABC";
PRINT "DEF";

```
ABCDEF
```

- If a semicolon or comma is not entered following the **<data to display>**, after the data is displayed, a carriage return and line feed is performed.
- If the line to display exceeds one screen line, the data is displayed continuously on the next line.
- When numeric values are displayed, one whitespace character is always inserted at the end. Also, one whitespace is inserted before all positive values, while a negative sign (-) is inserted before all negative values.

REMARK

See LOCATE, PRINT USING, SPC and TAB. Also see section 3.10.

- Displays a character string or numeric value in a specified format.

PRINT USING

SYNTAX	**PRINTΔUSINGΔ"<format string>" ;<data to display>**

	format string	Specifies the format in which to display the character string or numeric value.
	data to display	Specifies the character string or numeric value to be displayed.
EXAMPLE	**PRINT USING "###.##";123.56**	123.56 is displayed.

```
1 2 3.  5 6
```

	PRINT USING "& &","ABCD"	ABCD is displayed in four characters.

```
A B C D
```

DESCRIPTION

- The **PRINT USING** command displays data in a specified format.
- **<Format string>** (see the following pages) specifies the format in which to display the character string or numeric value.
- Data can be output to the console screen or the multi-purpose port.

[Setting the Display Format for Character Strings]

When using the **PRINT USING** command to display a character string, one of the following format characters must be used.

Format Character	Function	Example
"!"	The first character of the specified character string is displayed.	PRINT USING "!";"ABC" A OK
"& n characters of WS* &"	• The first (n + 2) characters of the specified character string are displayed.	PRINT USING "& &";"ABCD" ABCD OK
	• If no whitespace is specified in between the two ampersands (&), two characters are displayed.	PRINT USING "& &";"ABCD";"EFGH" ABCDEFGH OK
	• If the specified character string is longer than the format string, the extra characters are not displayed.	
	• The specified character string is displayed starting with the left side of the format string. Any extra characters in the format string are filled in with whitespace.	PRINT USING "& &";"AB";"CD" AB CD OK

*WS = whitespace.

[Setting the Display Format for Numeric Values]

When using the PRINT USING command to display a numeric value, one of the following format characters must be used.

Format Character	Function	Example
"#"	• The numeric value is displayed in each column position.	PRINT USING "####";123 123 OK
	• Numbers or whitespace are displayed in each column position designated in the format string.	
	• If the number of columns in the numeric value to be displayed is less than the number of columns in the format string, the numeric value is displayed right-justified within the format string, and any leftover columns at the left are filled in with whitespace.	PRINT USING "##";123 %123 OK
	• If the number of # characters is less than the number of columns in the numeric value, a percent symbol (%) is displayed at the head of the numeric value.	

Format Character	Function	Example
"."	• A decimal point is inserted. The decimal point can be inserted at any desired position in the format string. • If a format string follows the decimal point, the corresponding columns are definitely displayed. • Any unneeded columns following the decimal point are filled in with zeroes. • If the number of decimal places in the numeric value to be displayed is greater than the number of decimal places in the format string, the number is rounded off at the last decimal place that can be displayed. • If the number is rounded off in order to fit in the display area specified by the format string, a percent sign (%) is displayed at the head of the number.	PRINT USING "##. ##";.78 0.78 OK PRINT USING "##.##";78 78.00 OK PRINT USING "##.##";78.125 78.13 OK PRINT USING ".##";.999 %1.00 OK
"+"	If this character is specified at the beginning or end of the format string, the positive or negative sign is displayed before or after the numeric value depending on whether the value is positive or negative.	PRINT USING "+##.#";-68.9 -68.9 OK PRINT USING "+##.#";24 +24.0 OK PRINT USING "##.#+";-55 55.0- OK
"-"	If this character is specified at the end of the format string, when negative values are displayed, the negative sign is displayed following the value.	PRINT USING "##.#-";-70.1 70.1- OK
"***"	If these characters are specified at the beginning of the format string, any whitespace at the head of the numeric value is displayed as an asterisk (*) instead. "***" reserves a two column region just the same as "##".	PRINT USING "**#.#";12.39 *12.4 OK PRINT USING "**#.#";765.1 765.1 OK
"\\"	• If these characters are specified at the beginning of the format string, a "\" character is displayed just to the left of the numeric value. "\\" defines a two column region, but only one "\" character is displayed. • The "\\" characters cannot be specified for numbers displayed in exponential notation.	PRINT USING "\\###.#";456.7 \456.7 OK
"***\"	• If these characters are specified at the beginning of the format string, "**" characters are displayed to the left of the numeric value, and a "\" character is inserted just before the numeric value. • The "**\" characters reserve a three column region, but only one "\" character is displayed.	PRINT USING "**\##.##";2.34 ***\2.34 OK

Format Character	Function	Example
","	• If a comma is specified to the left of the decimal point, the number to the left of the decimal point is separated by a comma after every three columns. • The comma reserves a one column region. • If the comma is specified together with the exponential notation characters ("^^^^"), the comma is ignored.	PRINT USING "####,.##";1234.5 1.234.50 OK PRINT USING "####.##,";1234.5 1234.50, OK
"^^^^"	• If these characters are specified at the end of the format string, exponential notation is selected. • The "^^^^" characters reserve a region in which to display the "E+XX" portion of the number. • The significant figures are displayed from the left, and the exponent is then added at the end. • If the "+" character is not specified at the beginning, or the "+" or "-" characters are not specified at the end, a negative sign (-) or whitespace is displayed to the left of the decimal part of the number.	PRINT USING "##.##^^^^";234.56 2.35E+02 OK PRINT USING ".####^^^^";888888 .0889E+07 OK PRINT USING "+.##^^^^";123 +.12E+03 OK

POINT

If the number of columns specified is 25 or greater, an "Illegal function call" error occurs.

[Multi-Field]

Several format strings can be specified within one PRINT USING command. Each format string is separated by a non-format string character. Each format string corresponds to the numeric value or character string to be displayed based on their order of specification. If the number of format strings is less than the number of data items to be displayed, the format strings are re-used starting at the first format string. Thus, the format strings are "looped" until all data items have been displayed.

Example

PRINT USING "No. ##### AD51H & &";12345;"BASIC MANUAL"

No. 12345 AD51H BASIC MANUAL

OK

- Writes data to a sequential file.

PRINT #

SYNTAX	**PRINTΔ#<file number>, <data>**	
	file number	Specifies the file number assigned to a sequential file when it was opened with the OPEN command.
	data to display	Specifies the numeric value or character string data to be written to the file.
EXAMPLE	**PRINT #1, A**	The numeric value stored in variable A is written to the file with file number 1.
	PRINT #1, A$	The character string stored in variable A$ is written to the file with file number 2.

DESCRIPTION

- The **PRINT#** command writes data to a sequential file.
- Before executing the **PRINT#** command, a sequential file must be opened with the **OPEN** command.
- The **PRINT#** command writes data to a sequential file exactly the same way as the **PRINT** command displays data on the display screen. Therefore, the numeric values and character strings within **<data to display>** must be separated with a semicolon. If the data items are separated with a comma instead of a semicolon, the whitespace normally displayed between the items is written to the file.
- The **PRINT#** command does not display data on the screen.

REMARK

See OPEN, INPUT#, LINE INPUT#, CLOSE, PRINT, PRINT USING, and PRINT# USING.
Also see chapter 6.

- Writes data to a sequential file in a specified format.

PRINT # USING

SYNTAX	**PRINTΔ#<file number>, USINGΔ"<format string>";<data to display>**

	file number	Specifies the file number assigned to a sequential file when it was opened with the OPEN command.
	format string	Specifies the format in which to write the character string or numeric value.
	data to display	Specifies the numeric value or character string data to be written to the file.
EXAMPLE	**PRINT #1,USING "####& &";A;B$**	The "#" characters are filled in with the numeric value stored in variable A, and the "& &" is filled in with the character string stored in B$, and both are written to the file. A = 1234 B$ = "ABCD" Sequential File #1 1234ABCD
	PRINT #2,USING "\\###.#";C	The numeric value stored in variable C is written to the file with a "\" character attached to the beginning. C = 456.7 Sequential File #2 Δ\456.7

DESCRIPTION

- The **PRINT# USING** command writes data to a sequential file in a specified format.
- Before executing the **PRINT# USING** command, a sequential file must be opened with the **OPEN** command.
- When the **PRINT# USING** command is executed, the contents of **<data to display>** are written to the file according to the style specified with <format string>. Data is not displayed on the screen.

POINT

For information on how to set the at string for the PRINT# USING command, see the explanation of the PRINT USING command.

REMARK

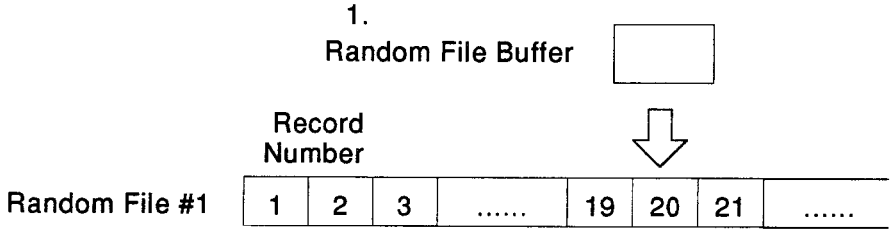
See PRINT, PRINT USING, and PRINT #. Also see chapter 6.

- Writes one record from the random file buffer to the random file.

PUT

SYNTAX	**PUTΔ#<file number> [,<record number>]**

	file number	Specifies the file number of the file to which data will be written.
	record number	Specifies the record to which the data will be written.
EXAMPLE	PUT #1, 20	Data is written to the 20th record of file number 1.



DESCRIPTION

- The **PUT** command writes the data from the random file buffer to the random file.
- If **<file number>** is omitted the record following the one specified in the previous **GET** or **PUT** command is written.
  If the **<record number>** is omitted in the first **PUT** command executed after the **OPEN** command, the record number is assumed to be one.
- When the **PUT** command is executed the data in the random file buffer is written to the specified 256 byte record.

REMARK

See OPEN, FIELD, GET, LSET and RSET. Also see chapter 6.

- Writes data to the AD51H buffer memory, common memory or an internal device (ED).

PUT MEMory

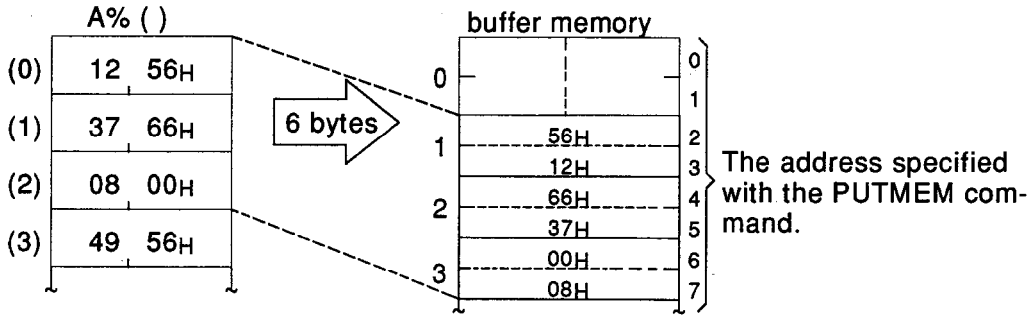
SYNTAX	**PUTMEMΔTOΔ<write destination>, <offset 1>, <FROM>Δ <data storage area>, <offset 2>, <number of bytes>**

write destination	Specifies the buffer memory or common memory head address, or the internal word device head device.
data storage area	Specifies the integer variable, integer array, character string variable or character string array in which the data to be written is stored.
offset 1	Specifies the offset (in byte units) from the head of the write destination.
offset 2	Specifies the offset (in byte units) from the head of the data storage area.
number of bytes	Specifies the length of the data to write in byte units.

EXAMPLE

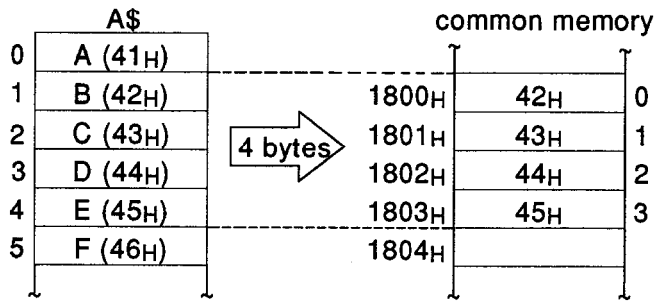
PUTMEM TO 0,2FROM A%( ),0,6

The data stored in A%(0) to A%(2) (6 bytes) is written to AD51H buffer memory 1 to 3.



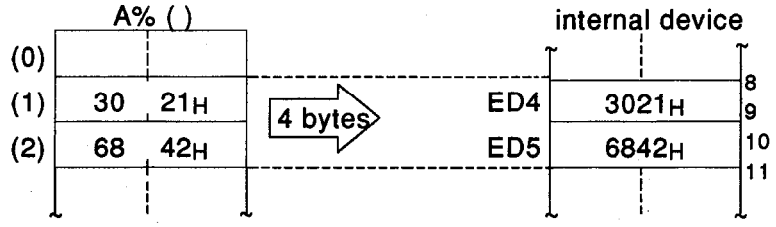
PUTMEN TO &H1800, 0 FROM A$,1,4

The 1st to 4th characters of A$ (4 bytes) are written to the AD51H common memory addresses &H1800h to &H1803.



PUTMEM TO W@(ED,2),4 FROM A%( ),2,4

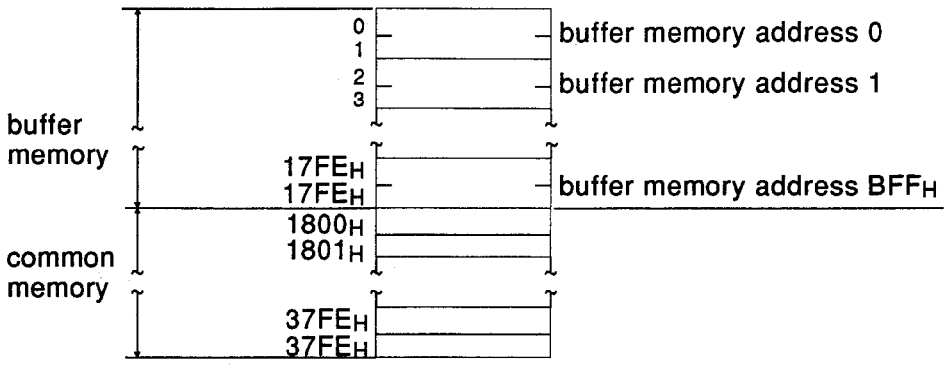
The data in A%(1) and A%(2) (4 bytes) is written to ED4 and ED5.



DESCRIPTION

- The **PUTMEM** command writes the data specified by **<data storage area>**, **<offset 2>** and **<number of bytes>** to the buffer memory, common memory or internal device (ED) area specified by **<write destination>** and **<offset 1>**.
- A buffer memory or common memory address, or an internal word device may be specified for **<write destination>**.
  Buffer memory and common memory addresses are organized as follows.
  The head address to write to is specified with **<write destination>**.

Specifying addresses with PUTMEM



POINT

As shown above, one buffer memory address uses 2 bytes. Therefore buffer memory addresses should be specified the following way with the PUTMEM command.

(buffer memory address specified with the PUTMEM command) = (AD51H buffer memory address) ×2

When writing to an internal word device, the head device is specified the same as when using special variable W@ (ED,n).

To use ED 100 as the head device → **W@ (ED, 100)**

To use ED 20 as the head device → **W@ (ED, 20)**

REMARK

For information on how to store the data to be written, see:
section 4.3 (buffer memory).
section 8.6 (common memory, internal devices).

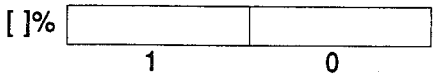
- The variable, array, etc. in which the data to be written is stored should be specified with **<data storage area>**. The **<offset>** and **<byte count>** parameters should be used to specify exactly what data within **<data storage area>** is to be written.

POINT

Even if the array variable to be used as the <data storage area> has less than ten elements, it must be defined with the DIM command. When the PUTMEM command is executed, only the portion of data equal to the array size as defined with the DIM command is written.

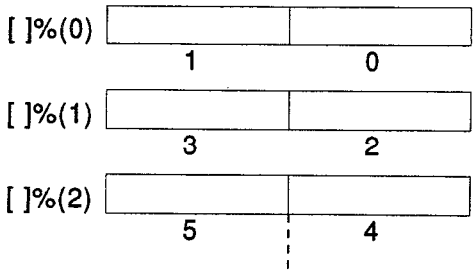
- When a character string variable or string array is used as the **<data storage area>** the code corresponding to each character is written to the write destination.
  <Offset 1> is used to specify an offset in byte units from the address specified in **<write destination>**.
- **<Offset 2>** is used to specify what address is to be the starting address to which data will be written. It is specified as an offset in byte units from the address specified in **<write destination>**.

Specifying an integer variable



An integer variable is composed of two bytes (16 bits). An offset of 0 specifies the lower byte; an offset of 1 specifies the upper byte.

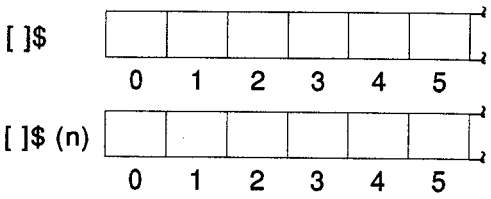
Specifying an integer array



Each element of an integer array is composed of two bytes (16 bits). The offset value is specified as follows.

- To specify the lower byte of array element number n:
  <offset> = n × 2
- To specify the upper byte of array element number n:
  <offset> = n × 2 + 1

Specifying a character string variable or a character string array



Each character within a string variable or array is one byte. To specify the nth character within a variable, the following equation is used:
<offset> = n - 1

- The length of the data to be written starting at the head position specified with **<offset 2>** is specified in byte units with **<number of bytes>**.

  If an integer variable or array is specified for **<data storage area>**, since each integer element (variable) can store two bytes of data, the storage capacity of n elements is described by the following equation:

  <number of bytes> = n × 2

  If a string variable or array is specified for **<data storage area>**, since each character can store one byte of data, the storage capacity of n characters is described by the following equation:

  <number of bytes> = n

- The position, size and type of data in buffer memory, common memory, etc. is managed by the user.
  When data is read from or written to buffer memory, common memory, etc. no type conversion is performed.

- When programs are running under multitasking, and it is possible that several programs may try to read or write to the same range of buffer memory, exclusive access management should be performed using the **ZRESERVE** and **ZRELEASE** commands.
  If a program tries to read or write using an area of common memory that is already being read from or written to, an error does not occur. Common memory may be simultaneously accessed.

PROGRAMMING EXAMPLE

```
10 'Write to the extended register (TASK 1)
20 ZODV 0                                    :'Output to the console
30 DIM A%(9)                                 :'Define an array
40 DEF ZEVENT 1                              :'Define an event
50 ZEVENT ENABLE 1                           :'Enable the event
60 A$="TASK 1 WRITING DATA"                  :'Define messages
70 B$="TASK 1 FINISHED WRITING DATA"
80 FOR I=1 TO 10                             :'Write the data
90  A%(I-1)=I
100 NEXT I
110 ZRESERVE 1                               :'Reserve resource number 1
120 LOCATE 0,2                               :'Specify a display position
130 PRINT A$                                 :'Display the message
140 PUTMEM TO W@(ED,0),0 FROM A%( ),0,20     :'Execute write
150 PRINT B$                                 :'Display the message
160 ZSIGNAL 1                                :'Signal the event
170 PRINT "DATA THAT WAS WRITTEN"            :'Display the data that was written
180 FOR I=1 TO 10
190  PRINT A%(I-1),
200 NEXT I
210 ZRELEASE 1                               :'Release resource number 1
220 END                                      :'End execution
```

```
10 'Read from the extended register (TASK 2)
20 ZODV 0                                    :'Output to the console
30 DIM B%(9)                                 :'Define an array
40 C$="TASK 2 READING DATA"                  :'Define messages
50 D$="TASK 2 FINISHED READING DATA"
60 FOR I=0 TO 9                              :'Store dummy data
70  B%(I)=0
80 NEXT I
90 ZWAIT EVENT 1                             :'Wait for the event to be signalled
100 ZRESERVE 1                               :'Reserve resource number 1
110 LOCATE 0,15                              :'Specify a display position
120 PRINT C$                                 :'Display the message
130 GETMEM TO B%( ),0 FROM W@(ED,0),0,20     :'Execute read
140 PRINT D$                                 :'Display the message
150 PRINT "DATA THAT WAS READ"               :'Display the data that was read
160 FOR I=1 TO 10
170  PRINT B%(I-1),
180 NEXT I
190 ZRELEASE 1                               :'Release resource number 1
200 END                                      :'End execution
```

REMARK

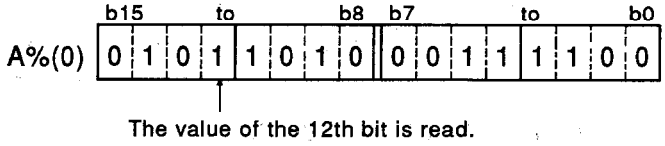
See GETMEM. Also see section 8.5.1.

- Reads one bit of data from a specified bit within an integer array variable.

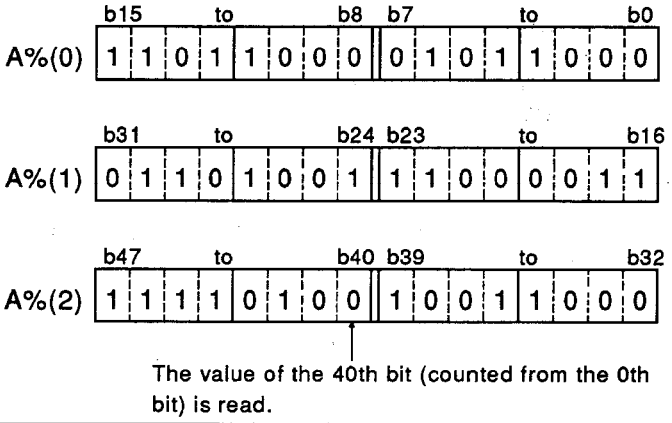
ReaD SET

SYNTAX	**RDSET (<integer expression>, <array variable>)**

	integer expression	Specifies from which bit within the specified integer array to read the data. Must be a value in the range -32768 to 65535.
	array variable	Specifies a one dimensional integer array from which data is to be read.
EXAMPLE	**B=RDSET(12,A%(0))**	The value of the 12th bit of A%(0) (counted from the 0th bit) is read into B.



	B=RDSET(40,A%(0))	The value of the 40th bit of A%(0) (counted from the 0th bit) is read into B.



DESCRIPTION

- The **RDSET** function reads the contents of the bit specified by **<integer expression>** from a one-dimensional integer array specified with **<array variable>**.
- The bit specified with **<integer expression>** is counted in order from the 0th bit of the specified array variable, each bit of higher array elements can be specified continuously as an offset from the 0th bit of the specified element. For example, if values in the range 0 to 15 are specified, each bit of the specified array variable can be accessed. However, if values in the range 16 to 31 are specified, each bit of the array element following the one specified can be accessed (see example). If the specified value exceeds the number of elements defined for the array with the DIM command, an **"Illegal function call"** error occurs.
- Only a one dimensional integer array variable may be specified for **<array variable>**. If any other type of variable is specified, an **"Illegal function call"** error occurs.

POINT

Integers that can be handled with 16 bits are in the range -32768 to 32767. The hexadecimal representation for these numbers is shown below. If values in the range -32767 to -1 (base 10) are specified, they are handled as if values in the range 32768 to 65535 had been specified.

Decimal	Hexadecimal	Binary
65535 to 32768	FFFFH to 8000H	1111 1111 1111 1111 to 1000 0000 0000 0000
32767 to 0	7FFFH to 0000H	0111 1111 1111 1111 to 0000 0000 0000 0000
-1 to -32768	FFFFH to 8000H	1111 1111 1111 1111 to 1000 0000 0000 0000

These values are identical.

PROGRAMMING EXAMPLE

```
10 'Read one bit of data from a specified bit.
20 DIM A%(0)                                  :'Define an array
30 A%(0)=&H5F5F
40 FOR I=0 TO 15                              :'Loop I from 0 to 15
50 A=RDSET(I,A%(0))                           :'Read
60 PRINT A;                                   :'Display
70 NEXT I

RUN
1  1  1  1  1  0  1  0  1  1  1  1  1  0  1  0
OK
```

REMARK

See WTSET.

- Reads the values defined with the DATA command and stores them to variables.

READ

SYNTAX	**READ∆<variable name> [,<variable name>] . . .**

	variable name	Specifies the variable into which the values defined with the DATA command will be read.
EXAMPLE	**READ A, B, C$**	Numeric values from the DATA line are read into A and B, and a character string value is read from the DATA line into C$.

```
READ A , B , C$
DATA 100 , 200 , "ABCD"
```

DESCRIPTION

- The **READ** command reads the values defined with the **DATA** command and stores them to variables.
- The **READ** command must be used together with the **DATA** command.
- The variable names specified in the **READ** command are in one to one correspondence with the values defined in the **DATA** command.
- The variable names specified in the **READ** command may represent numeric or character string type variables, but the type of variable specified must correspond with the type of value to be read. If the variable type and the value type do not correspond, a **"Syntax error"** occurs at the **DATA** command, and execution is halted.
- One **READ** command can refer to one or several **DATA** lines in order.
- If the number of variables specified in the **READ** commands exceeds the number of values defined in the **DATA** commands, the error message **"Out of data"** is displayed.
- If there are no more **READ** commands to be executed, any unread values remaining in the **DATA** commands are ignored.
  The **RESTORE** command can be used to re-read the values defined in the **DATA** commands from the beginning, or from a specified location.

REMARK

See DATA and RESTORE. Also see section 3.5.2 .

- Places a comment in the program.

REMark

SYNTAX	**REMΔ[<comment>]**	
	comment	The comment is entered.
EXAMPLE	**REM **AD51H****	Indicates that the program is related to the AD51H.
	or	
	'AD51H****	

DESCRIPTION

- The **REM** command places a comment in the program. The comment has no effect on program execution. When a program list is performed, the comment is displayed as it was entered.
- A **GOTO** or **GOSUB** command may branch execution to a **REM** command, but execution actually begins with the line following the **REM** command.
- An apostrophe can be used as a substitute for the **REM** command. Comments that begin with an apostrophe can be placed on the same line with other commands. However, other commands cannot be entered continuously on the same line following the comment. The entire section of the line from the apostrophe to the end of the line is considered to be one comment.

REMARK

See section 2.5.

- Renumbers a program's line numbers.

RENUMber

SYNTAX	**RENUMΔ[<new starting line number>] [, [<old starting line number>] [, <interval>]]**

	new starting line number	Specifies the line number which is to become the first line number when the lines are renumbered.
	old starting line number	Specifies a line number within the program to be renumbered.
	interval	Specifies the interval between each of the new line numbers.
EXAMPLE	**RENUM 500, 100, 5**	The lines currently starting at line 100 are renumbered so that the new line numbers start at 500 and proceed with intervals of five.

DESCRIPTION

- The **RENUM** command renumbers the line numbers in a program.
- Any value in the range 1 to 65529 may be specified for **<new starting line number>** and **<old starting line number>**.
- If **<new starting line number>** is omitted, the value 10 is assumed.
- If **<old starting line number>** is omitted, renumbering begins at the start of the current program.
- **<Interval>** specifies the difference between the line numbers of consecutive lines. If it is omitted, the value 10 is assumed.
- The **RENUM** command automatically changes the line numbers specified in the **GOTO**, **GOSUB**, **IF...THEN**, **IF...GOTO**, **ON...GOTO**, **ON...GOSUB**, **RESTORE** and **RESUME** commands, and the **ERL** function to correspond with the new line numbers.
  The line numbers specified with the **CHAIN** command are not changed.
  If a nonexisting line number is specified in any of these commands, the error message **"Undefined line xxxxx in yyyyy"** is displayed during renumbering.
  xxxxx indicates the specified line that did not exist, and yyyyy indicates the new line number of the command which specified the nonexisting line number.
  In this case, the line number that did not exist is not changed.

POINT

The RENUM command cannot be used to change the order of program lines (for example, if program lines numbered 10, 20 and 30 exist, the command RENUM 15, 30 is not valid).

(incorrect) PROGRAM

```
10
20   RENUM 15, 30
30   ---------------------> ×
```

(correct) PROGRAM PROGRAM

```
10                    30
20   RENUM 30, 10     40
30                    50
```

Also, line numbers greater than 65529 cannot be generated. If the renumber procedure causes such lines to be created, an "Illegal function call" error occurs.

- Allows the values defined with the DATA command to be re-read by the READ command starting at a specified line number.

RESTORE

SYNTAX	**RESTOREΔ[<line number>]**	
	line number	Specifies the line number of the DATA command whose values are to be read by the next READ command.
EXAMPLE	**RESTORE 60**	Reading starting will begin with the data defined by the DATA command in line 60.

DESCRIPTION

- The **RESTORE** command is used to specify from which **DATA** command the next **READ** command will read values. The **DATA** command is specified by line number.
- If the **<line number>** is omitted, the next **READ** command to be executed following the **RESTORE** command will read data from the first **DATA** command in the program.
- If the **<line number>** is specified, the next **READ** command to be executed following the **RESTORE** command will read data from the first value of the **DATA** command on the specified line number.
- A label may be specified for **<line number>**.

PROGRAMMING EXAMPLE

```
10  'Specify the location from which the READ command will read data using the RESTORE
    command.
20  RESTORE 200                              :'Read data starting with line 200

40  READ A$,B$,C$:PRINT A$,B$,C$             :'Read and display data
60  RESTORE 100                              :'Read data starting with line 100

80  READ A$,B$,C$:PRINT A$,B$,C$             :'Read and display data
90  END
100  DATA U,V,W
200  DATA X,Y,Z

RUN
X            Y            Z
U            V            W
OK
```

REMARK

See READ and DATA. Also see section 3.5.2.

- Returns execution to the main procedure following execution of an error management procedure defined with the ON ERROR GOTO command.

RESUME

SYNTAX	**RESUMEΔ[<execution return position>]**

execution return position — Specifies the location where execution of the main procedure will be restarted following completion of the error processing routine.

- If nothing is specified — Execution is restarted at the command which originally caused the error.
- If NEXT is specified — Execution is restarted at the command following the command that originally caused the error.
- If a line number (or label) is specified — Execution is restarted at the specified line number or label.

EXAMPLE

RESUME	Error processing is completed, and program execution is restarted at the location where the error was caused.
RESUME NEXT	Error processing is completed, and program execution is restarted at the command following the location where the error was caused.
RESUME 150	Error processing is completed, and program execution is restarted at line 150.

DESCRIPTION

- The **RESUME** command ends the error management routine called by the **ON ERROR GOTO** command and restarts program execution.
- The following types of **RESUME** commands can be used to select where program execution should be restarted.

  (1) RESUME
  Program execution is restarted from the command which caused the error.

  (2) RESUME NEXT
  Program execution is restarted from the command following that which caused the error.

  (3) RESUME <line number>
  Program execution is restarted from the specified <line number>.

PROGRAMMING EXAMPLE

```
 10 ON ERROR GOTO 100
 20 A=0
 30 PRINT 100/A
 40 PRINT "END"
 50 END
100 REM  ERROR PROCESSING
110 A=1
120 RESUME [      ]
```

An error occurs because this becomes 100÷0.

line number specified	NEXT specified	nothing specified
⇩	⇩	⇩
Execute from the specified line number.	Execute from the command following the one that caused the error.	Execute from the command that caused the error.

- Returns as its value the substring starting a specified number of characters from the right of the specified string and ending at the end of the string.

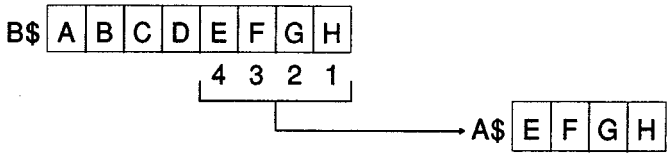
RIGHT$

SYNTAX	**RIGHT$Δ(<character string expression>,<numeric expression>)**

character string expression	The character string from which a substring will be read.
numeric expression	The number of characters to be read.

EXAMPLE

A$=RIGHT$(B$,4) The last four characters of B$ are read out and stored in A$.



DESCRIPTION

- The **RIGHT$** function returns as its value a substring which begins a specified number of characters from the right of a character string and ends at the right side of the string.
- **<Numeric expression>** specifies the length of the substring to be read.
- If the value of **<numeric expression>** is zero, the empty string is returned.
  If the value of **<numeric expression>** exceeds 255, an **"Illegal function call"** error occurs.
- If the value of **<numeric expression>** is equal to or greater than the length of **<character string expression>**, the entire string is read out.

PROGRAMMING EXAMPLE

```
10  'Read the rightmost seven characters from a character string
20  A$ = "Sequence Program"                    :'Define a character string
30  B$=RIGHT$(A$,7)                            :'Read the rightmost seven characters
50  PRINT "The rightmost seven characters --->";B$ :'Display the substring
60  END

RUN
The rightmost seven characters --->Program
OK
```

REMARK

See LEFT$, MID$ (version 1) and MID$ (version 2). Also see section 3.13.2.

- Returns a random number.

RaNDom

SYNTAX	**RND (<numeric expression>)**

	numeric expression	Depending on whether this value is negative, 0, or positive, the number generated is different.
EXAMPLE	**A = RND(1)**	The value generated by RND(1) is stored in A.
	B = RND(0)	A random number in the sequence last used is generated and assigned to B.
	C = RND(-1)	A new sequence is created, a random number is generated, and assigned to C.

DESCRIPTION

- The **RND** function returns a random number between 0 and 1.
- The random number generated depends on the value of **<numeric expression>**.
  If **<numeric expression>** is positive, a new random number is generated.
  If **<numeric expression>** is zero, the random number last generated is re-generated.
  If **<numeric expression>** is negative, a new random number is generated.

PROGRAMMING EXAMPLE

```
10  'Program example to generate random numbers
20  DIM A(10)                                  :'Define an array
30  PRINT "RND(1)"
40  FOR I=1 TO 10                              :'Loop I from 1 to 10
50   A(I)=RND(1)                               :'Generate a random number with RND(1)
60   PRINT A(I)                                :'Display
70  NEXT I

RUN
RND(1)
 .286546
 .784621
 .137119
 .226544
 .876763
 .8574
 .567958
 .364675
 .0333645
OK
```

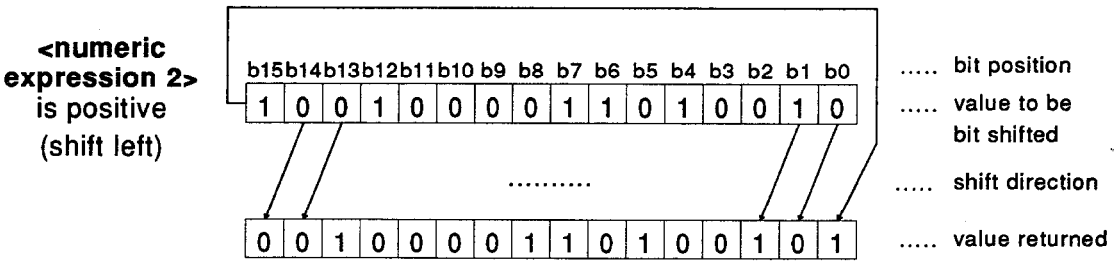
- Performs a bit rotation on a specified value's memory contents and returns the bit rotated value.

ROTation

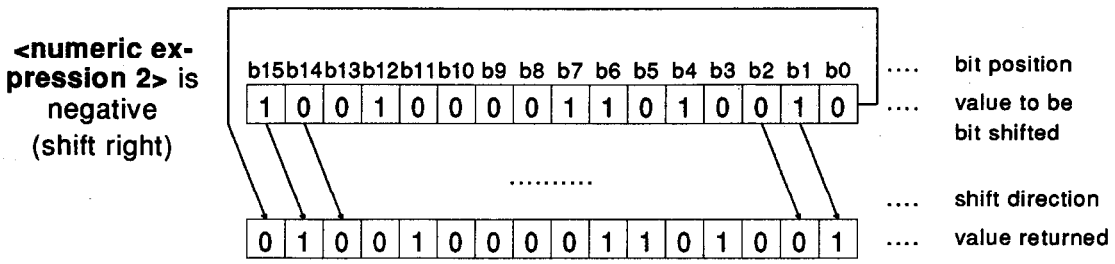
SYNTAX	**ROTΔ (<expression 1>, <expression 2>)**	
	numeric expression 1	Specifies the integer value which is to be bit rotated.
	numeric expression 2	Specifies the shift direction and number of bits as an integer.
EXAMPLE	**A%=ROT(B%, C%)**	The value of B% is bit shifted the amount specified in C% and store to A%.
	D%=ROT(E%, 12)	The value of E% is shifted 12 bits to the left and stored it in D%.
	F%=ROT(G%, -4)	The value of G% is shifted 4 bits to the right and stored it in F%.

DESCRIPTION

- The **ROT** command shifts the memory contents of the value specified with **<numeric expression 1>** the amount specified by **<numeric expression 2>** and returns the shifted value.
- **<Numeric value 2>** specifies the bit rotation of the value specified in **<numeric value 1>**.
  If the value of **<numeric expression 2>** is positive, the bit rotation direction is left, the contents of the highest bit (the sign bit) are moved to the lowest bit.



If the value of <numeric expression 2> is negative, the bit rotation direction is right, the contents of the lowest bit are moved to the highest bit.



- If anything but integer data is specified for **<numeric expression 1>** or **<numeric expression 2>**, the data is converted to integer type before being processed.
- If **<numeric expression 1>** is not in the range -32768 to 32767, an error occurs.
- If **<numeric expression 2>** is not in the range -16 to 16, an error occurs.

PROGRAMMING EXAMPLE

```
10 'The value of A% is bit shifted.
20 A%=&H55FF
30 B%=1                                    :'Specify the value to be bit shifted
40 C%=-1
50 D%=ROT(A%, B%)                          :'Bit shift
60 E%=ROT(A%, C%)
70 PRINT "A%=";A%;"shifted one bit to the left -----> D%=";D%:'Display
80 PRINT "A%=";A%;"shifted one bit to the right -----> E%=";E%

RUN
A%= 22015 shifted one bit to the left -----> D%= 21506
A%= 22015 shifted one bit to the right -----> E%= 21761
OK
```

REMARK

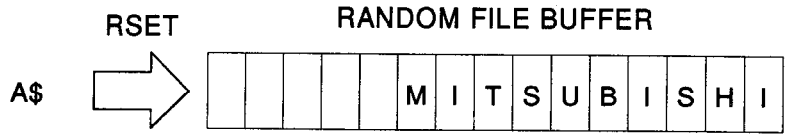
See SHA and SHT.

- Moves data from memory to the random file buffer in preparation for the PUT command.

Right SET

SYNTAX	**RSETΔ<character string variable>=<character string expression>**

	character string variable	A character variable that was defined with the FIELD command is specified.
	character string expression	Specifies the character string data to be written.
EXAMPLE	**RSET A$="MITSUBISHI"**	Stores the string "MITSUBISHI" right justified to the variable

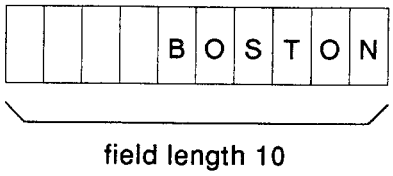


DESCRIPTION

- The **RSET** command moves data from memory to the random file buffer in preparation for the **PUT** command.
- In order to execute this function, a file must first be opened as a random file.
- Before the **RSET** command can be executed, character variables must be defined with the **FIELD** command.
- If the length of the character expression is shorter than the field length assigned to the character variable with the **FIELD** command, the **RSET** command stores the character expression right-justified within the field. The unused portion at the left of the field is filled in with whitespace.

Example

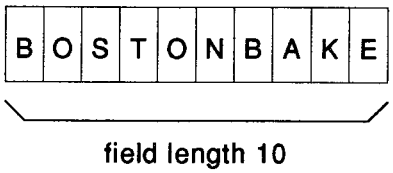
field length: 10
character expression: BOSTON



- If the length of the character expression is longer than the field length assigned to the character variable with the **FIELD** command, the right section of the the character expression is cut off.

  Example

  field length: 10
  character expression: BOSTONBAKEDBEANS



- Before a numeric value is moved into the random file buffer, it must be converted into a character string using the **MKI$**, **MKS$** or **MKD$** functions.

REMARK

See FIELD, PUT, GET and LSET. Also see chapter 6.

- Begins execution of the program currently stored in the program area.

RUN

SYNTAX	**RUNΔ[<line number>]**	
	line number	Specifies the line number where execution will begin.
EXAMPLE	**RUN**	Execution is started from the line with the smallest line number.

DESCRIPTION

- The **RUN (version 1)** command begins execution of the program currently stored in the program area.
- If **<line number>** is specified, execution begins at that line number.
- If **<line number>** is omitted, execution begins at the line with the smallest line number. When execution is complete, BASIC enters the "waiting for command input" condition.
- The **RUN** command closes all open files and clears all memory.

REMARK

See CONT and STOP. Also see chapter 3.

- Loads a program from a memory card, floppy disk or hard disk and executes it.

RUN

SYNTAX	**RUNΔ"[<drive number>:] [<system name>\] <file name>"**

	drive number	Specifies the memory card, floppy disk or hard disk where the program is stored.
	system name	Specifies the system name where the program to be loaded is stored.
	file name	Specifies the file name of the program to be loaded.
EXAMPLE	**RUN "0:TEST\NO1.BAS"**	The file named "NO1.BAS" stored under system name "TEST" on the memory card in memory card slot [1] is loaded and executed.

DESCRIPTION

- The **RUN (version 2)** command loads a program from a memory card, floppy disk or hard disk and executes it.
- **<Drive number>** specifies the memory card,floppy disk or hard disk from which to load the program. The following values may be specified:

  AD51H Memory Card [1] 0

  AD51H Memory Card [2] 1

  PC/AT FDD 2

  PC/AT HDD[C] 3
- <System name> and **<file name>** refer to the names specified when the program was stored to the memory card, or floppy disk using the SAVE command.
- The RUN command closes all open files and clears all memory before loading and executing the program.

REMARK

See section 3.3.

- Stores a program to a memory card, floppy disk or hard disk.

SAVE program

SYNTAX	**SAVEΔ"[<drive number>:] [<system name>\] <file name>"**

	drive number	Specifies the memory card, floppy disk or hard disk where the program is to be stored.
	system name	Specifies the system name under which the program is to be stored.
	file name	Specifies the file name of the program to be stored.
EXAMPLE	**SAVE "0:TEST\No1.BAS"**	The program currently in memory is stored to the file named "No1.BAS" under system name "TEST" on the memory card in memory card slot [1].

DESCRIPTION

- The **SAVE** command stores the program currently in memory to a memory card, floppy disk or hard disk.
- **<Drive number>** specifies the memory card, floppy disk or hard disk to which to store the program. The following values may be specified:

  ON-LINE PROGRAMMING

  AD51H Memory Card [1] 0

  AD51H Memory Card [2] 1

  PC/AT FDD 2

  PC/AT HDD[C] 3

  OFF-LINE PROGRAMINNG

  PC/AT FDD 1

  PC/AT HDD[C] 3

- <System name> specifies the system name under which the program is to be stored.
  If the specified system name does not exist, it is created.
  If **<system name>** is omitted, the file is stored as a file with no system name.
  Note that a system name cannot be created within another system name.
- **<File name>** specifies the name and file extension of the file in which the program is to be stored. If the file extension is omitted, the new file name will have no extension.

POINT

WARNING: If the file specified by <drive number>:<system name> \ <file name> already exists, it is overwritten without confirmation.

REMARK

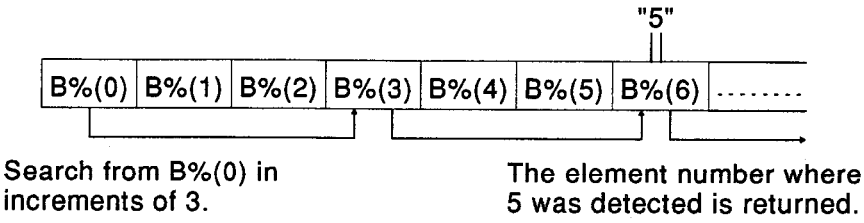
See LOAD. Also see section 3.3.

- Searches for a specified value within a specified array, and returns the position of the element where the value was found.

SEARCH

SYNTAX	**SEARCHΔ([<array variable name>,<search value>[,<starting element position>][,<step value>])**

	array variable name	Specifies the array variable to be searched (one dimensional integer type only).
	search value	Specifies the integer value which is to be searched for within the array variable.
	starting element position	Specifies from which element to begin the search.
	step value	Specifies the increment in which to search the elements.
EXAMPLE	**A=SEARCH(B%,5,0,3)**	Array variable B% is searched for the value five, starting with element B%(0) and proceeding in increments of three.



DESCRIPTION

- The **SEARCH** function searches the array specified with **<array variable>** for the value specified with **<sarch value>**, and returns as its value of the number of the element where the value is first found.
  The search begins at the element specified by **<starting element position>** and ends at the last element of the specified array.
  If the specified value cannot be found within the array, the value -1 is returned.
- If the specified array variable is not a one dimensional integer array, an **"Illegal function call"** error occurs.
- The array variable specified for **<array variable name>** in the SEARCH command must be defined with the **DIM** command before the **SEARCH** command is executed.
- If the value specified for **<sarch value>** is a real number, it is converted to integer before execution.
- If **<starting element position>** is omitted, the search begins at the first element of the array.
- If the **<step value>** is omitted, an increment of one is assumed, and all array elements are searched.

PROGRAMMING EXAMPLE

```
10  'Search for a value within an array.
20  DIM A%(10)                                  :'Define an array
30  FOR I=0 TO 10                               :'Loop I from 0 to 10
40   A%(I)=I+3                                  :'Store values in the array
50   PRINT "A%(";I;")=";A%(I)
60  NEXT I
70  B=SEARCH(A%,6,0,3)                          :'Search for 6
80  PRINT "6 is in A%(";B;")"                   :'Display results
90  END

RUM
A%(0)=3
A%(1)=4
A%(2)=5
A%(3)=6
A%(4)=7
A%(5)=8
A%(6)=9
A%(7)=10
A%(8)=11
A%(9)=12
A%(10)=13
6 is in A%(3)
OK
```

- Returns the sign of a numeric value.

SiGN

SYNTAX	**SGN(<numedric expression>)**	
	numeric expression	Specifies the numeric value whose sign is to be searched.
EXAMPLE	**A=SGN(B%)**	If the value of B% is positive, 1 is stored to A, if it is zero, zero is stored to A, if it is negative, -1 is stored to A.

DESCRIPTION

- The **SGN** function returns the sign of a numeric value.
- If **<numeric expression>** is positive, the function returns 1, if it is zero, the function returns zero, if it is negative, the function returns -1.

PROGRAMMING EXAMPLE

```
10  'Return the sign of a numeric value.
20  A=123                                  :'Define a numeric value
30  B=0
40  C=-123
50  PRINT "SGN(A)=;SGN(A)                  :'Display
60  PRINT "SGN(B)=;SGN(B)
70  PRINT "SGN(C)=;SGN(C)

RUN
SGN(A)= 1
SGN(B)= 0
SGN(C)=-1
OK
```

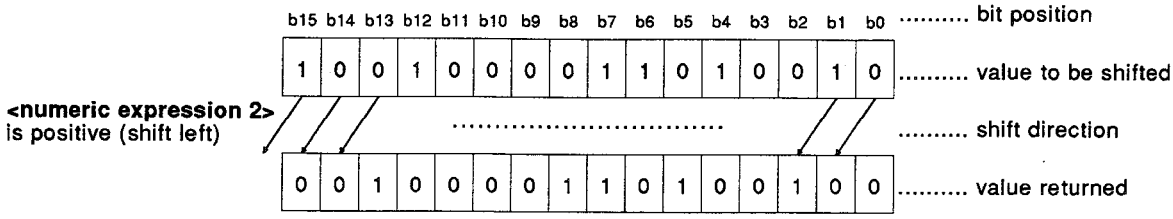
- Performs an arithmetic shift on a specified value's memory contents and returns the new value.

SHift Arithmetic

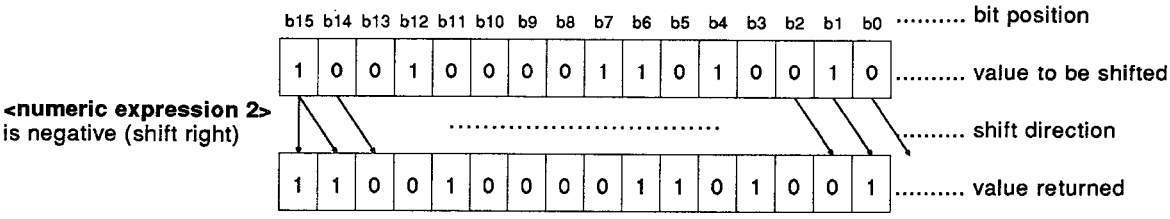
SYNTAX	**SHA(<numeric expression 1>,<numeric expression 2>)**	
	numeric expression 1	Specifies the integer value which is to be shifted.
	numeric expression 2	Specifies the shift direction and number of bits as an integer.
EXAMPLE	**A%=SHA(B%,C%)**	The value of B% is arithmetic shifted the amount specified in C% and stored to A%.
	D%=SHAS(E%,12)	The value of E% is arithmetic shifted 12 bits to the left and stored in D%.
	F%=SHA(G%,-4)	The value of G% is arithmetic shifted 4 bits to the right and stored in F%.

DESCRIPTION

- The **SHA** function arithmetic shifts the memory contents of the value specified with **<numeric expression 1>** the amount specified by **<numeric expression 2>** and returns the shifted value.
- **<Numeric expression 2>** specifies the arithmetic shift of the value specified in **<numeric expression 1>**.
  If the value of **<numeric expression 2>** is positive, the shift direction is left, and the lowest bits are turned OFF in order, becoming zero. Therefore, if 16 is specified as the value of **<numeric expression 2>**, the function returns zero.



If the value of **<numeric expression 2>** is negative, the shift direction is right, and the contents of the highest bit remains unchanged. Therefore, depending on the value of the highest bit, if -16 is specified for **<numeric expression 2>**, the function returns -1 or zero.



- If anything but integer data is specified for **<numeric expression 1>** or **<numeric expression 2>**, the data is converted to integer type before being processed.
- If **<numeric expression 1>** is not in the range -32768 to 32767, an error occurs.
- If **<numeric expression 2>** is not in the range -16 to 16, an error occurs.

PROGRAMMING EXAMPLE

```
10  'The value of A% is arithmetic shifted.
20  A%=&H5555
30  B%=1                                    :'Specify the value to be shifted
40  C%=-1
50  D%=SHA(A%,B%)                           :'Arithmetic shift
60  E%=SHA(A%,C%)
70  PRINT "A%=";A%;"arithmetic shifted once to the left---> D%=";D%:'Display
80  PRINT "A%=";A%;"arithmetic shifted once to the right--> E%=";E%


RUN
A%= 21845 arithmetic shifted once to the left---> D%= 21846
A%= 21845 arithmetic shifted once to the right--> E%= 10922
OK
```

REMARK

See ROT and SHT.

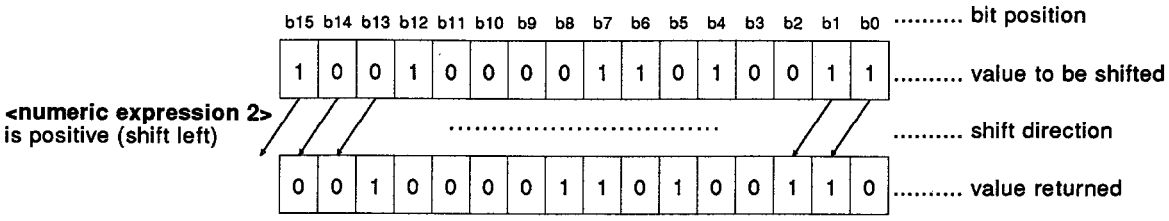
- Performs a logical shift on a specified value's memory contents and returns the new value.

SHifT logical

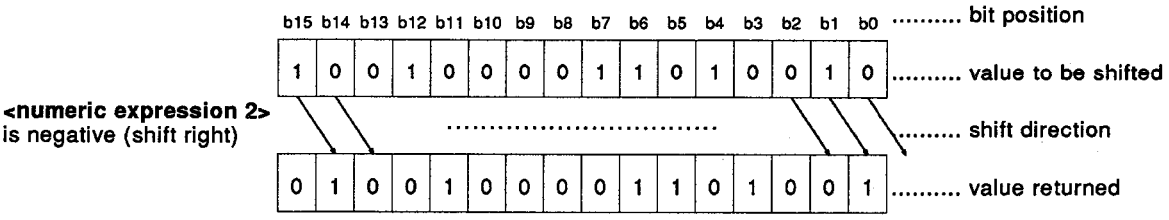
SYNTAX	**SHT(<numeric expression 1>,<numeric expression 2>)**	
	numeric expression 1	Specifies the integer value which is to be shifted.
	numeric expression 2	Specifies the shift direction and number of bits as an integer.
EXAMPLE	**A%=SHT(B%,C%)**	The value of B% is logically shifted the amount specified in C% and stored to A%.
	D%=SHT(E%,12)	The value of E% is logically shifted 12 bits to the left and stored in D%.
	F%=SHT(G%,-4)	The value of G% is logically shifted 4 bits to the right and stored in F%.

DESCRIPTION

- The **SHT** function logical shifts the memory contents of the value specified with **<numeric expression 1>** the amount specified by **<numeric expression 2>** and returns the shifted value.
- **<Numeric expression 2>** specifies the logical shift of the value specified in **<numeric expression 1>**.
  If the value of **<numeric expression 2>** is positive, the shift direction is left, and the lowest bits are turned OFF in order, becoming zero. Therefore, if 16 is specified as the value of **<numeric expression 2>**, the function returns zero.



If the value of **<numeric expression 2>** is negative, the shift direction is right, and the highest bits are turned OFF in order, becoming zero. If -16 is specified for **<numeric expression 2>**, the function returns zero.



- If anything but integer data is specified for **<numeric expression 1>** or **<numeric expression 2>**, the data is converted to integer type before being processed.
- If **<numeric expression 1>** is not in the range -32768 to 32767, an error occurs.
- If **<numeric expression 2>** is not in the range -16 to 16, an error occurs.

PROGRAMMING EXAMPLE

```
10  'The value of A% is logically shifted.
20  A%=&H5555
30  B%=1                                    :'Specify the value to be shifted
40  C%=-1
50  D%=SHT(A%,B%)                           :'Logical shift
60  E%=SHT(A%,C%)
70  PRINT "A%=";A%;"logically shifted once to the left---> D%=";D%:'Display
80  PRINT "A%=";A%;"logically shifted once to the right--> E%=";E%

RUN
A%= 21845 logically shifted once to the left---> D%=-21846
A%= 21845 logically shifted once to the right--> E%= 10922
OK
```

REMARK

See ROT and SHA.

- Returns the value of the trigonometric SINE function.

SINe

SYNTAX	**SIN (<numeric expression>)**	
	numeric expression	Specifies a numeric value or numeric variable in radians.
EXAMPLE	**A=SIN(3.14159/180*60)**	The angle measure 60 degrees is converted to radians, and its sine is calculated. The result is stored in A.

DESCRIPTION

- The **SIN** function returns the sine of the value specified in radians by **<numeric expression>**.
- The **<numeric expression>** can be any type of number, but the SIN function performs calculations in single precision.

PROGRAMMING EXAMPLE

```
10  'Return SIN 45°
20  A%=(3.141592653#*45)/180                :'Convert 45° to radians
30  B=SIN(A)                                :'Return SIN 45°
40  PRINT "45°      =";A;"radians"
50  PRINT "sin 45° =";";B
60  END

RUN
45°      = .785398 radians
sin 45° = .707107
OK
```

REMARK

See COS and TAN.

- Returns a specified number of whitespace characters.

SPACE$

SYNTAX	**SPACE$ (<numeric expression>)**	
	numeric expression	Specifies the length of the whitespace string.
EXAMPLE	**A$=SPACE$(10)**	Ten whitespace (space) characters are stored in A$.

DESCRIPTION

- The **SPACE$** function returns a whitespace string of a specified length.
- **<Numeric expression>** specifies the length of the whitespace string.
- The value specified for **<numeric expression>** must be between 0 and 255. If it exceeds 255, an **"Illegal function call"** error occurs.
- If zero is specified for **<numeric value>** the function returns the empty string (" ").

PROGRAMMING EXAMPLE

```
10  'Slide the starting position of each line to the right and display
20  FOR I=1 TO 5                              :'Loop I from 1 to 5
30   A$="AD51H"                               :'Define the character string
40   A$=SPACE$(I*5)+A$                        :'Add (I*5) whitespace characters to A$
50   PRINT A$                                 :'Display
60  NEXT I
70  END

RUN
     AD51H
          AD51H
               AD51H
                    AD51H
                         AD51H
OK
```

REMARK

See SPC and STRING$.

- Returns a specified number of whitespace characters.

SPaCe

SYNTAX	**SPC (<numeric expression>)**	
	numeric expression	Specifies the number of whitespace characters.
EXAMPLE	**PRINT SPC(1);A$**	One whitespace character is displayed, then the contents of A$ are displayed.

DESCRIPTION

- The **SPC** function returns a specified number of whitespace characters.
- The **SPC** function cannot be used with any commands other than **PRINT** and **LPRINT**. Further, it cannot be used independently.
- Output can be sent to a terminal or PC/AT attached to the communication port designated by the **ZODV** command, or to a printer attached to the communication port designated by the **ZLDV** command. For information on the output communication port, see the **ZODV** command.
- The value specified for **<numeric expression>** must be between 0 and 255. If it exceeds 255, an **"Illegal function call"** error occurs.

PROGRAMMING EXAMPLE

```
10  'Slide the starting position of each line to the right and display
20  FOR I=1 TO 5                          :'Loop I from 1 to 5
30   A$="BASIC"                           :'Define the character string
50   PRINT SPC(I*5);A$                    :'Display (I*5) whitespace characters and A$
60  NEXT I
70  END

RUN
     BASIC
          BASIC
               BASIC
                    BASIC
                         BASIC
OK
```

REMARK

See ZODV, ZLDV, SPACE$ and TAB.

- Returns the value of the square root.

SQuare Root

SYNTAX	**SQR (<numeric expression>)**	
	numeric expression	Specifies the numeric value whose square root is to be calculated.
EXAMPLE	**A=SQR(50)**	The square root of 50 (=7.071067...) is stored to A.

DESCRIPTION

- The **SQR** function returns the square root.
- The value of **<numeric expression>** must be zero or a positive number.
- The **SQR** function performs calculations in single precision.

PROGRAMMING EXAMPLE

```
10 'Calculate the square root
20 DIM A(10)                                  :'Define an array
30 PRINT "A(I)=","SQR(A(I))="
40 FOR I=1 TO 10                              :'Calculate and display the square root
50  A(I)=SQR(I)
60 PRINT I,
70 PRINT A(I)
80 NEXT I

RUN
A(I)=            SQR(A(I))=
 1               1
 2               1.41421
 3               1.73205
 4               2
 5               2.23607
 6               2.44949
 7               2.64575
 8               2.82843
 9               3
 10              3.16228
OK
```

- In edit mode, temporarily stops program execution.
- In execute mode, stops program execution, and causes the program to enter the "stopped" condition.

STOP

SYNTAX	**STOP**	
EXAMPLE	**STOP**	Program execution is temporarily stopped, and BASIC enters "waiting for command input" condition.

DESCRIPTION

EDIT MODE

- The **STOP** command temporarily stops program execution, and causes BASIC to enter the "waiting for command input" condition.
- The **STOP** command can be used anywhere in the program.
- When the **STOP** command is executed, the following message is displayed:

  Break in nnnnn nnnnn indicates the line number.
- Unlike the **END** command, the **STOP** command does not close any files.
- When execution is stopped by the **STOP** command, and BASIC enters the "waiting for command input" condition, if the **CONT** command is executed, program execution is restarted.

  However, if the program is changed after it is stopped, program execution cannot be restarted with the **CONT** command.

EXECUTE MODE

- The STOP command stops program execution, forcing the program into the "stopped" condition.
- If a debugger is in use, the debugger commands **TRUN** and **TCONTINUE** can restart program execution.
- The **STOP** command can be used anywhere in the program.

POINT

In execute mode, if the END command is omitted at the end of the program, it is the same as if a STOP command had been executed.

REMARK

See CONT and RUN. Also see chapter 10.

- Converts a numeric value (assumed to be base 10) to a character string.

STRing$

SYNTAX	**STR$ (<numeric expression>)**	
	numeric expression	Specifies the value to be converted to a character string.
EXAMPLE	**A$=STR$(-1.602)**	-1.602 is converted to the character string "-1.602".
	B$=STR$(3.12E08)	3.12E08 is converted to the character string "Δ3.12E08".

DESCRIPTION

- The **STR$** function returns a base 10 representation of **<numeric expression>** as a character string. Only numeric values or variables may be specified. **<Numeric expression>** can be integer, real (single or double precision) or fixed point type; all numeric types are valid.
- The length of the string returned is equal to the number of digits in the numeric value, plus one column for the sign. The sign column for positive values is filled in with whitespace.
- If the number of digits in the specified number exceeds 16 for double precision values, or 6 for single precision values, exponential notation is used.
- The **VAL** function has the opposite effect from the **STR$** function. Also, the **OCT$** and **HEX$** functions are used to convert integer data to octal or hexadecimal character string representations.

PROGRAMMING EXAMPLE

```
10 'Convert numeric values to character strings
20 A=123                    :'Define numeric values
30 B=456
40 C=1.23456E+07
50 A$=STR$(A)               :'Convert to character strings
60 B$=STR$(B)
70 C$=STR$(C)
80 PRINT "A     =";A        :'Display
90 PRINT "B     =";B
100 PRINT "C     =":C
110 PRINT "A+B   =";A+B
120 PRINT "A$+B$ =";A$+B$
130 PRINT "C$    =";C$
140 END
```

```
RUN
A      = 123
B      = 456
C      = 1.23456E+07
A+B    = 579
A$+B$  = 123 456
C$     = 1.23456E+07
OK
```

REMARK

See HEX$, OCT$ and VAL.

- Returns a string containing a specified number of a specified character.

STRING$

SYNTAX	**STRING$(<numeric expression 1>, <character expression>)** **STRING$(<numeric expression 1>, <numeric expression 2>)**

	numeric expression 1	Specifies the number of characters to return.
	character expression	Specifies the character to return.
	numeric expression 2	Specifies the character code of the character to return.
EXAMPLE	**A$=STRING$(5,"M")**	The character "M" is repeated five times and stored to A$.

A$

M	M	M	M	M

	B$=STRING$(3,&H41)	The character with character code &H41 ("A") is repeated three times and stored to B$.

B$

A	A	A

DESCRIPTION

- The **STRING$** function returns a string containing a specified character repeated a specified number of times.
- **<Numeric expression 1>** specifies how many times to repeat the character.
- Only the first character of **<character expression>** is valid. If the character is specified with <numeric expression 2> the character returned by the specified code is used. The value must be in the range 0 to 255. If the value exceeds 255, an **"Illegal function call"** error occurs.

PROGRAMMING EXAMPLE

```
10 'Display the "*" character according to the value of variable A.
20 FOR I=0 TO 5                                  :'Loop I from 0 to 5
30  A=2^I                                        :'Calculate the value of A
50  A$=STRING$(A,"*")                            :'Store the "*" character the number of
                                                  times indicated by the calculation in line
                                                  50
60  PRINT A$:'Display
70 NEXT I
80 END

RUN
*
**
****
********
****************
********************************
OK
```

REMARK

See CHR$, SPACE$ and SPC.

- Aborts BASIC control and returns to AD51H system mode.
- Aborts off-line programming and returns to the main menu screen.

SYSTEM

SYNTAX	**SYSTEM**	
EXAMPLE	**SYSTEM**	BASIC operation is aborted.

DESCRIPTION

- BASIC (the system that manages BASIC program editing and execution) operation is aborted, and control is returned to the previous mode.

ON-LINE PROGRAMMING

[edit mode]
Control is returned to **AD51H system mode**.

[execute mode]
All tasks are stopped and control is returned to AD51H system mode.
In addition, an Illegal function call error occurs.
When the **SYSTEM** command is used to end BASIC operation, all files are closed.
The programs in memory remain as stored, but all variables and data are cleared.
For information on AD51H system mode, see the AD51H/AD51H-S3 User's Manual (On-Line Programming, Debugging volume).

OFF-LINE PROGRAMMING

Control is returened to SW[ ]IX-AD51HPE system main menu screen.
For information on SW[ ]IX-AD51HPE system, see the Operating Manual.

- Exchanges the values of two variables.

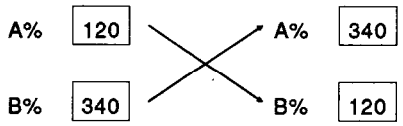
SWAP

SYNTAX	**SWAPΔ<variable 1>, <variable 2>**

variable 1 — Specifies the variables whose values are to be exchanged.

variable 2

EXAMPLE **SWAP A%,B%** The values of A% and B% are exchanged.



DESCRIPTION

- The **SWAP** command exchanges the value of the variable specified by **<variable 1>** with the value of the variable specified by **<variable 2>**.
- Any type of variable may be specified for **<variable 1>** and **<variable 2>**. However, if both variables are not the same type, a **"Type mismatch"** error occurs.

PROGRAMMING EXAMPLE

```
10 'Exchange the contents of variables A and B.
20 INPUT "A=";A                              :'Input A
30 INPUT "B=";B                              :'Input B
40 PRINT "A=";A, "B=";B                      :'Display before exchange
50 SWAP A,B                                  :'Exchange
60 PRINT "Values have been exchanged."
70 PRINT "A=";A, "B=";B                      :'Display after exchange
80 END

RUN
A=? 1
B=? 2
A= 1            B= 2
Values have been exchanged.
A= 2            B= 1
OK
```

REMARK

See BSWAP.

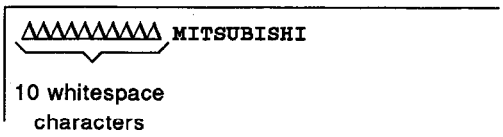
- Advance the current display position to a specified location.

TAbulate

SYNTAX	**TAB(<numeric expression>)**

numeric expression — Specifies the display location offset from the left edge.

EXAMPLE — **PRINT TAB(10); "MITSUBISHI"** "MITSUBISHI" is printed from the 11th character position.



DESCRIPTION

- The **TAB** function displays whitespace characters from the current display position up until the specified display position on the screen. If the current display position exceeds the specified one, a carriage return is performed.
- The **TAB** function can be used with the following screens:

  Console Screen

- The **TAB** function cannot be used with any commands other than **PRINT** and **LPRINT**.

PROGRAMMING EXAMPLE

```
10 'Display a character string from the position specified with the TAB function.
20 A$="AD51H"                                :'Define character strings
30 B$="BASIC"
40 PRINT TAB(5);A$;TAB(15);B$                :'Display from the 5th and 15th columns
50 END

RUN
    AD51H    BASIC
OK
```

REMARK

See SPC and LOCATE. Also see section 3.10.1 .

- Returns the value of the trigonometric function tangent.

TANgent

SYNTAX	**TAN(<numeric expression>)**	
	numeric expression	A numeric value or variable is specified in radian units.
EXAMPLE	**A=TAN(3.14159/180*45)**	The angle measure 45 degrees is converted to radians, and the tangent of this value is stored to A.

DESCRIPTION

- The **TAN** function returns the tangent of the value specified in radian ((π/180) × degrees) units with **<numeric expression>**.
- The numeric expression can be specified in any numeric type, but calculations are normally carried out in single precision.
- If the following values are specified for value, a **"Division by zero"** error occurs.

  ((2n + 1) × π)/2 **n = 0, 1, 2, 3, ...**
  π = 3.1415926

PROGRAMMING EXAMPLE

```
10 'Return tan 45°
20 A+(3.141592653#*45)/180                    :'Convert 45° to radians
30 B=TAN(A)                                   :'Return tan 45°
40 PRINT "45°      =";A;"radians"
50 PRINT "tan 45° =";B
60 END

RUN
45°      = .785398 radians
tan 45° = 1
OK
```

REMARK

See SIN and COS.

- Sets the time of the PC CPU clock.
- Reads the time from the PC CPU clock.

TIME$

SYNTAX	**TIME$= "<hours>:<munites>:<seconds>"** **TIME$**

	hours	A character string in the range "00" to "23" specifying the hour.
	minutes	A character string in the range "00" to "59" specifying the minute.
	seconds	A character string in the range "00" to "59" specifying the second.
EXAMPLE	**TIME$ = "11:57:30"**	The PC CPU clock is set to 11:57:30.
	A$ = TIME$	The contents of the PC CPU clock are read out and stored in A$.

DESCRIPTION

- The **TIME$** function sets and reads out the contents of the PC CPU clock.
- To set the time, the following syntax is used:

  TIME$= "hours:minutes:seconds"

  1) A colon must be used to separate hours, minutes and seconds.
  2) Hours must be set in 24-hour style, in the range 0 to 23.
  3) If any of the settings is in the range 0 to 9, an extra zero must be added at the front. The settings should be specified as 00 to 09.
  4) All three settings, hours, minutes and seconds, must be entered.
- When reading the time from the clock, the same syntax may be used as when setting the clock.

POINTS

(1) The TIME$ function is only valid for PC CPUs that have a clock function. The TIME$ function cannot be used with PC CPUs that have no clock function (A[ ], A3H, A0J2(H) CPUs).

(2) When using the TIME$ function to read the hour, minute and second, before reading, dummy data must be stored to the variable which is to be used, as shown below.
If dummy data is not stored, an error occurs.

[ ]$ = SPACE$ (14)
[ ]$(n) = SPACE$ (14)

PROGRAMMING EXAMPLE

```
10 'Set and read the time.
30 A$="13:23:01"                          :'Set the time
40 TIME$=A$                               :'Write the time
60 B$=SPACE$(8)                           :'Store dummy data
70 B$=TIME$                               :'Read the time
80 PRINT "Clock contents-->";B$           :'Confirm the time that was written
90 END

RUN
Clock contents-->13:23:01
OK
```

REMARK

For information on the precision of the PC CPU clock see the User's Manual for the CPU in use.

See DATE$.

- Resets the tracing of a program established with the TRON command.

TRace OFF

SYNTAX	**TROFF**

EXAMPLE	**TROFF**	Resets trace execution.

DESCRIPTION

- The **TROFF** command is used together with the **TRON** command to debug a program.
- When the **TROFF** command is executed, tracing is ended.
- The **TROFF** command can be executed in both direct mode and program mode.
- When any of the following commands are executed, tracing is ended, just as if the **TROFF** command had been executed.

  NEW
  RUN "file name"
  LOAD
  CHAIN
  MERGE

REMARK

See TRON. Also see chapter 10.

- Traces a program.

TRace ON

SYNTAX	**TRON**

EXAMPLE	TRON	Executes tracing.

DESCRIPTION

- The **TRON** command is used together with the **TROFF** command to debug a program.
- When the **TRON** command is executed, during program execution, the line number of each command that is executed is displayed in parentheses.
  The line number display can only be directed to the console (the output device designated by the **ZODV** command is ignored).
- The **TRON** command can be executed in both direct mode and program mode.
  When any of the following commands are executed, tracing is ended.

  TROFF
  NEW
  RUN "file name"
  LOAD
  CHAIN
  MERGE

- Screen display can be paused by pressing the [Ctrl]+[S] key. Pressing any key restarts the screen display.

REMARK

See TROFF. Also see chapter 10.

Returns the value represented by a character string.

VALue

SYNTAX	VAL (<character string expression>)

	character string expression	Specifies the character string to be converted to a numeric value.
EXAMPLE	A=VAL("123")	The numeric value represented by the character string "123" is stored to A.

DESCRIPTION

- The VAL function returns the value represented by a character string.

Decimal (base 10) Character Strings

Only whitespace, +, -, or numeric characters may be specified as the first character.
All characters starting with the second character must be numbers.
If whitespace, linefeed or tab characters are included in the string, they are ignored. Also, if a character other than one which represents a numeric value is included, all characters following it are ignored.

Example

PRINT VAL ("–Δ1Δ2Δ3") ..... -123
PRINT VAL ("Δ12A3") Δ12

Octal (base 8) Character Strings

The first two characters must be "&O" which indicates an octal number.
All characters starting with the third character must be numbers.
If whitespace, linefeed or tab characters are included after the first two characters of the string, they are ignored. Also, if a character other than one which represents a numeric value is included, all characters following it are ignored.

Example

Octal Number

PRINT VAL(&01Δ2Δ3") Δ83 (123)
PRINT VAL("&012A3") Δ10 (12)

Hexadecimal (base 16) Character Strings

The first two characters must be "&H" which indicates a hexadecimal number.
All characters starting with the third character must be numbers or "A" to "F".

If a character other than a number or "A" to "F" is included following the second character of the string, all characters following it are ignored.

Example Hexadecimal Number

PRINT VAL("&H1AΔ2C") ..... **Δ26 (1A)**
PRINT VAL("&H1A2!C") ..... **Δ418 (1A2)**

- If the first or second characters of the character string are anything but the standard defined characters, the function returns zero.
- The **VAL** function performs calculations in double precision.

PROGRAMMING EXAMPLE

```
10 'Convert a numeric value with the VAL command.
20 A$="1234"                    :'Define character strings
30 B$="12.34"
40 C$="1234"
50 D$="ABC"
60 E$="&H4F
70 F$="&H4F 56"
80 G$=E$+"12"
90 PRINT VAL(A$)                :'Convert the string "1234" to the numeric value 1234
100 PRINT VAL(B$)               :'Convert the string "12.34" to the numeric value 12.34
110 PRINT VAL(C$)               :'Ignoring whitespace, convert the string "1234" to the
                                 numeric value 1234
120 PRINT VAL(D$)               :'The string does not represent a numeric value, so
                                 return zero
130 PRINT VAL(E$)               :'Convert the hexadecimal number 4F to the numeric
                                 value 79
140 PRINT VAL(F$)               :'Since whitespace follows 4F, convert only the
                                 hexadecimal number 4F to the numeric value 79
150 PRINT VAL(G$)               :'Convert the hexadecimal number 4F12 to the numeric
                                 value 20242

RUN
 1234
 12.34
 1234
 0
 79
 79
 20242
OK
```

REMARK

See STR$, HEX$ and OCT$.

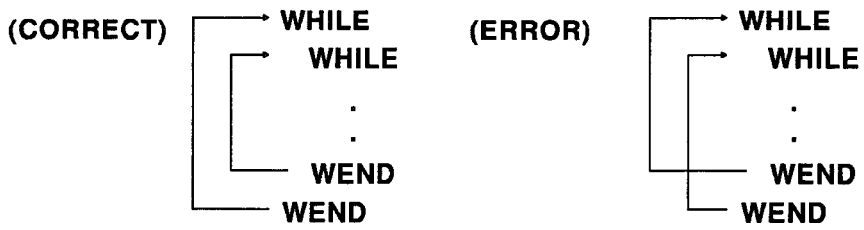
- Repeats the block of commands between the WHILE and WEND commands as long as a specified condition is met.

WHILE~While END

SYNTAX	**WHILE∆<expression>** **WEND**	
	expression	Specifies the condition under which the commands between the WHILE~WEND commands will be repeatedly executed.
EXAMPLE	**WHILE K>5** ↓ **WEND**	The WHILE~WEND block is repeatedly executed until the value of K is more than 5.

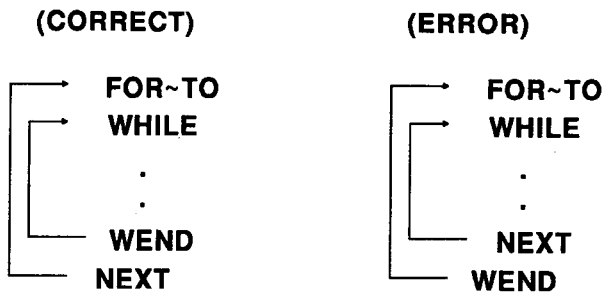
DESCRIPTION

- The **WHILE~WEND** command is used to repeatedly execute a specified procedure based on a condition. While the condition specified with is **<expression>** met, the commands up until the **WEND** command are repeated.
- The **WHILE** and **WEND** commands together make up one loop. Unlike the **FOR~NEXT**command, instead of specifying the number of times to repeat,the condition under which repetition will continue is specified.
- While the value of **<expression>** is TRUE (not 0), all commands between the **WHILE** and the **WEND** are executed, and then execution is returned to the **WHILE** command. **<Expression>** is re-evaluated, and if it is FALSE (0), execution jumps to the command following the **WEND** command.
- If a **WHILE~WEND** loop is used inside the error management procedure, and a **GOTO** or **RESUME** command is executed inside the loop, the loop is aborted.
- The line number of a line that contains a **WHILE~WEND** command can be specified as the jump destination for **GOTO** and similar commands.
- A **WHILE~WEND** loop can be included within another **WHILE~WEND** loop.
- When using nested **WHILE~WEND** loops, do not confuse the correspondence between **WHILE~WEND** commands.



POINT

When using a FOR~NEXT loop together with a WHILE~WEND loop, be sure not to mismatch the commands.



PROGRAMMING EXAMPLE

```
10 'Repeat a calculation according to a specified condition
20 I=1                                    :'Define the initial value
30 WHILE I<10                             :'Loop until I reaches 10
40  PRINT "I=";I                          :'Display I
50  I=I+1                                 :'Add 1 to I
60 WEND
70 PRINT "I has reached 10"               :'Display results
80 END

RUN
I= 1
I= 2
I= 3
I= 4
I= 5
I= 6
I= 7
I= 8
I= 9
I has reached 10
OK
```

REMARK

See FOR~NEXT. Also see section 3.6.4.

- Sets the printer width.

WIDTH

SYNTAX	**WIDTHΔLPRINTΔ<output width>**	
	output width	Specifies the output line width in the range 30 to 255.
EXAMPLE	**WIDTH LPRINT 100**	The printer output line width is set to 100.

DESCRIPTION

- The **WIDTH** command sets the width (the maximum number of characters per line) of data printed on the printer designated with the **ZLDV** command.
- After the **WIDTH** command is executed, when data longer than <
- output width> is printed, the characters up until the specified **<output width>** are printed, then a carriage return is performed, and the rest of the data is printed.
- If 255 is specified for **<output width>**, the output width is considered infinite, and the carriage return code is not sent to the printer.
- When BASIC is first started, the printer width is set to 80 characters.

PROGRAMMING EXAMPLE

```
10 'Set the output width and output to the printer
20 A$=MITSUBISHIAD51H,ABCDEFGHIJKLMNOPQRSTUVWXYZ,1234567890
   :'Define a character string
30 WIDTH LPRINT 50                          :'Set the output width to 50
40 LPRINT A$                                :'Output to the printer
50 WIDTH LPRINT 80                          :'Set the output width to 80
60 LPRINT A$                                :'Output to the printer
70 END

MITSUBISHIAD51H,ABCDEFGHIJKLMNOPQRSTUVWXYZ,1234567890
MITSUBISHIAD51H,ABCDEFGHIJKLMNOPQRSTUVWXYZ,1234567890
```

REMARK

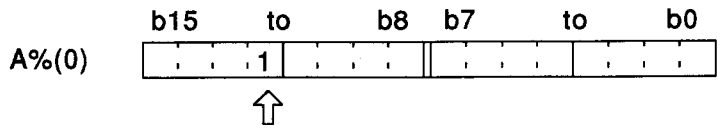
See LFILES, LLIST, LPRINT, LPRINT USING, PRINT #, PRINT # USING and ZLDV.

- Writes a zero or a one to a specified bit within an integer array variable.

WriTe SET

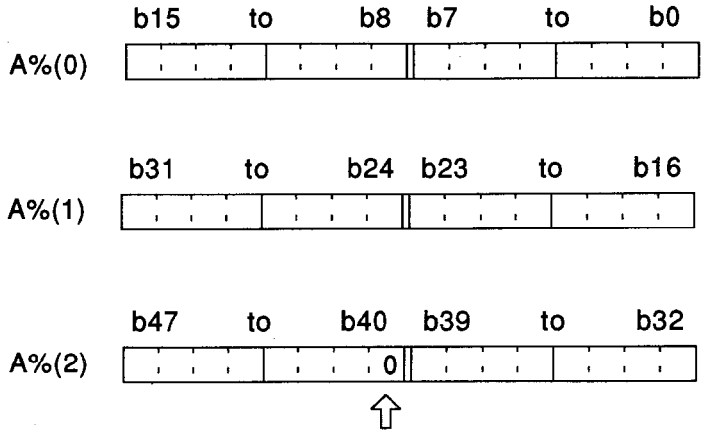
SYNTAX	**WTSETΔ(<integer expression 1>, <integer expression 2>, <array variable>)**

	integer expression 1	Specifies to which bit within the specified integer array to write the data. Must be a value in the range 0 to 65535.
	integer expression 2	Specifies the data (0 or 1) to write.
	array variable	Specifies a one dimensional integer array to which data is to be written.
EXAMPLE	**WTEST 12,1,A%(0)**	The value of the 12th bit of A%(0) (counted from the 0th bit) is set to 1.



The value of the 12th bit is set to 1.

WTEST 40,0,A%(0) The value of the 40th bit of A%(0) (counted from the 0th bit) is set to 0.



The value of the 40th bit (counted from the 0th bit) is set.

DESCRIPTION

- The **WTSET** command writes the data specified by **<integer expression 2>** (0 or 1) to the bit specified by **<integer expression 1>** in a one-dimensional integer array specified with **<array variable>**.

- The bit specified with **<integer expression 1>** is counted in order from the 0th bit of the specified array variable. Therefore, each bit of higher array elements can be specified continuously as an offset from the 0th bit of the specified element.
  If the specified value exceeds the number of elements defined for the array with the **DIM** command, an **"Illegal function call"** error occurs.

- The data to be written is set with **<integer expression 2>**.
  Only "0" or "1" may be specified.
  If any value other than "0" or "1" is specified, an **"Illegal function call"** error occurs.

- Only a one dimensional integer **<array variable>** may be specified for array variable.
  If any other type of variable is specified, an **"Illegal function call"** error occurs.

POINT

Integers that can be handled with 16 bits are in the range -32768 to 32767. The hexadecimal representation for these numbers is shown below.
If values in the range -32767 to -1 (base 10) are specified, they are handled as if values in the range 32768 to 65535 had been specified.

Decimal	Hexadecimal	Binary
65535 to 32768	FFFFH to 8000H	1111 1111 1111 1111 to 1000 0000 0000 0000
32767 to 0	7FFFH to 8000H	0111 1111 1111 1111 to 0000 0000 0000 0000
-1 to -32768	FFFFH to 8000H	1111 1111 1111 1111 to 1000 0000 0000 0000

These values are identical.

PROGRAMMING EXAMPLE

```
10  'Write one bit of data to a specified bit.
20  DIM A%(0)                                :'Define an array
30  FOR I=0 TO 14 STEP 2                     :'Write a "1" to every other bit
40  WTSET I,1,A%(0)
50  NEXT I
60  FOR I=1 TO 15 STEP 2                     :'Write a "0" to every other bit
70  WTSET I,0,A%(0)
80  NEXT I
90  FOR I=0 TO 15                            :'Read out
100  A=RDSET(I,A%(0))
110  PRINT A;                                :'Display
120  NEXT I

RUN
 1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0
OK
```

REMARK

See RDSET.

- Returns the BASIC task number area number of the program currently being written or executed.

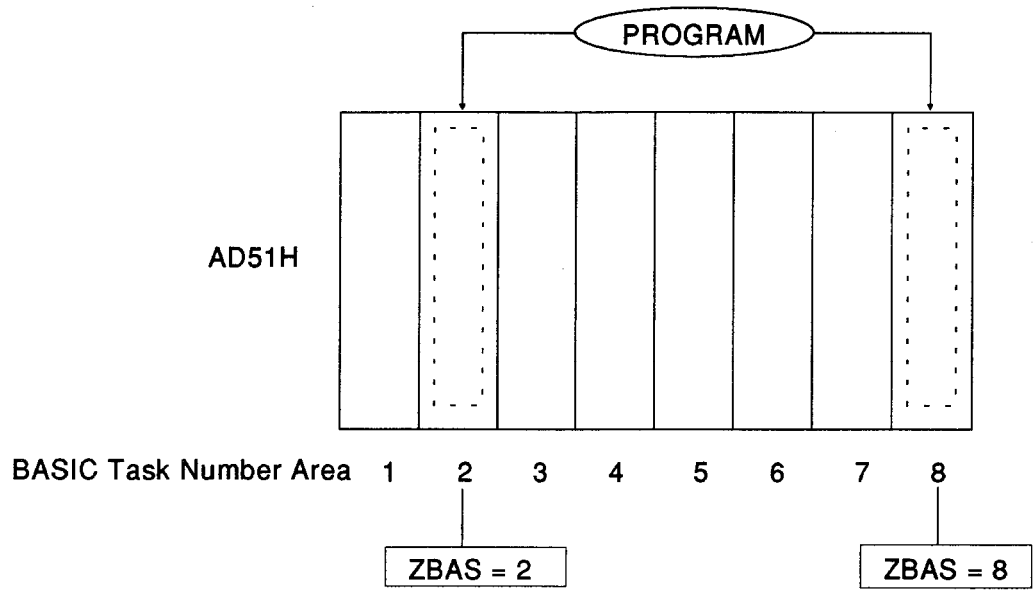
Z-BASic

SYNTAX	**ZBAS**

EXAMPLE **A=ZBAS** The BASIC number of the program currently being created or executed is stored in A.

DESCRIPTION

- The **ZBAS** function indicates the BASIC task number area number within the AD51H where the program currently being created or executed exists.
- Up to eight programs can exist on the AD51H, and each program is assigned a BASIC ask number area number. The ZBAS function returns this BASIC task number area number.



- Closes the communication channel of a communication port that was used for communication with an external device.

Z CLOSE

SYNTAX	**ZCLOSEΔ[#<channel number>], [#<channel number>, ...]**

	channel number	Specifies the communication port to be closed.
EXAMPLE	**ZCLOSE #1, #2**	CH1 (RS-232C) and CH2 (RS-232C) are closed.
	ZCLOSE	All open communication ports in the program are closed.

DESCRIPTION

- The **ZCLOSE** command closes a communication channel that was opened with the **ZOPEN** command and ends communication.
- If **<channel number>** is not specified, all open communication channels in the program are closed.
- Multiple **<channel number>** can be specified, but they must be separated with commas.
- Even if the **ZCLOSE** command is executed on a channel that is not open, an error does not occur.

REMARK

See ZOPEN. Also see chapter 7.

- Sets communication information and reads the status of an open communication port.

Z-CoNTroL

SYNTAX	**ZCNTL△#<channel number>, 0, <control table>**

	channel number	Specifies the communication port which is to be set, or whose status is to be read.
	control table	Specifies the data to be set, or the contents that were read.
EXAMPLE	**ZCNTL #1,0,A%( )**	Reads the condition of or sets data for CH1 (RS-232C) depending on the contents of integer array A%.

DESCRIPTION

- The **ZCNTL** command can perform the following settings on, or read the status of an open communication port.
- The contents of the process are defined with procedure codes in the control table. The procedure codes are shown below.

Procedure Code	Procedure Details	RS-232C	RS-422	PARALLEL
16	Set communication parameters	○	○	×
17	Read communication parameters	○	○	×
18	Set communication control parameters	○	○	×
19	Read communication control parameters	○	○	×
22	Set the break character	○	○	×
23	Read the break character	○	○	×
32	Set the RS and ER signals ON/OFF	○	○	×
33	Read the ON/OFF condition of the CS, DR, RS, ER and CD signals	○	○	×
64	Read the cause of a receive error	○	○	×
80	Read the size of the receive buffer and the number of characters	○	○	×
128	Read the printer status	×	×	○
136	Send the initialize signal to the printer	×	×	○

- **<Channel number>** specifies which AD51H communication port to use. The channel numbers and corresponding communication ports are shown below:

Channel Number	Communication Port
1	CH1 (RS-232C)
2	CH2 (RS-232C)
3	CH3 (RS-422)
4	CH4 (PARALLEL)

  The port to which settings will be made or from which the status will be read must be opened with the **ZOPEN** command prior to executing the **ZCNTL** command. If the port is not open, an error occurs.

- An integer array variable is used as the **<control table>** to specify the procedure code that indicates what kind of procedure is to be performed, and to specify any parameters that are necessary for the procedure. The **<control table>** is different depending on what kind of procedure is being performed, so please follow the instructions for each procedure code.

POINT

Even if the number of elements used by the <control table> is less than 10, the array used as the <control table> must be defined with the DIM command. If it is not defined with the DIM command, when the ZCNTL command is executed, an error occurs. (Normally, if the number of elements to be used is fewer than 10, an array need not be defined with the DIM command.)

Procedure Code 16 **Set communication parameters**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Baud Rate	Specifies the baud rate.
[ ]%(2)	Character Length/ Parity Bit	Specifies the character length in the lower byte, and the parity bit in the upper byte.
[ ]%(3)	Stop Bit	Specifies the stop bit in the lower byte. The upper byte is ignored.

Procedure Code 17 **Read out communication parameter settings**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Baud Rate	Reads out the baud rate.
[ ]%(2)	Character Length/ Parity Bit	Reads out the character length in the lower byte, and the parity bit in the upper byte.
[ ]%(3)	Stop Bit	Reads out the stop bit in the lower byte. The upper byte is ignored.

- The communication parameters of an open communication channel can be set or read out.
- The **procedure code** is 16 when setting communication parameters, and 17 when reading them.
  The other items are discussed below:

  Changing the set values ..... All items are set with the control table, using the same method as the **ZOPEN** command's control table specifications. For details, see the **ZOPEN** command. (All items must be set, individual items cannot be set separately.)

  Reading the setting contents ... The currently set values are stored in the control table element positions (1) to (3).
  For information on the meaning of each setting, see the **ZOPEN** command.

POINT

When the communication parameters are set, the receive buffer is completely cleared, and the condition of the control signals are all initialized.

PROGRAMMING EXAMPLE

```
10  'Set the communication parameters of a communication channel
20  DIM TBL1%(2),TBL2%(3),TBL3%(3),B%(1)      :'Define arrays
30  A%=1                                      :'Communication channel CH1 (RS-232C)
40  TBL1%(0)=4800                             :'Set baud rate
50  TBL1%(1)=&H107                            :'Set character length and parity bit
60  TBL1%(2)=&H1                              :'Set stop bit
70  ZOPEN #A%,TBL1%( )                        :'Open the communication channel
80  TBL2%(0)=16                               :'Specify set communication parameters
90  TBL2%(1)=9600                             :'Set baud rate
100  TBL2%(2)=&H208                           :'Set character length and parity bit
110  TBL2%(3)=&H1                             :'Set stop bit
120  ZCNTL #A%,0,TBL2%( )                     :'Execute setting communication parameters
130  TBL3%(0)=17                              :'Specify read communication parameters
140  ZCNTL #A%,0,TBL3%( )                     :'Execute read communication parameters
150  PRINT "baud rate=";TBL3%(1)              :'Display baud rate
160  A$=RIGHT$("0000"+HEX$(TBL3%(2)),4)       :'Convert TBL%3(2) to a character string
170  L$=RIGHT$(A$,2)                          :'Read the lower byte
180  H$=LEFT$(A$,2)                           :'Read the upper byte
190  B%(0)=VAL("&H"+L$)                       :'Convert the lower byte to a numeric value
200  B%(1)=VAL("&H"+H$)                       :'Convert the upper byte to a numeric value
210  PRINT "charactor length=";B%(0)          :'Display character length
220  PRINT "parity bit=";B%(1)                :'Display parity bit
230  PRINT "stop bit=";B%(3)                  :'Display stop bit
240  ZCNTL #A%                                :'Close communication channel
250  END
```

Procedure Code 18 **Set communication control parameters**

Control Table Contents

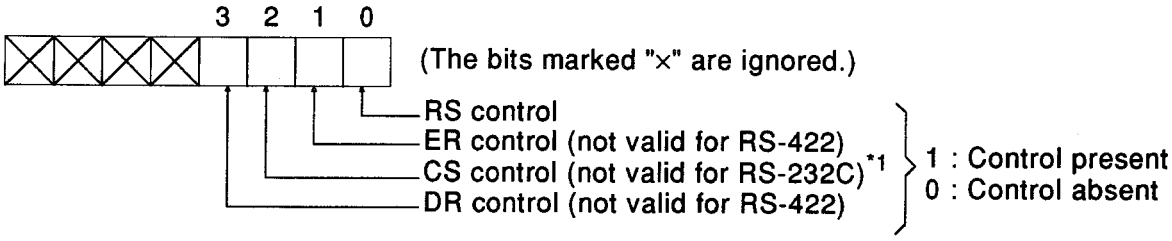
Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Signal Control, DC Control	The lower byte specifies the signal control format, and the upper byte specifies the DC control format.
[ ]%(2)	Dummy Data	It is not necessary to set this element. (Used by the system as a work area.)
[ ]%(3)	Receive Buffer	Specifies the size of the receive buffer.

Procedure Code 19 **Read communication control parameter settings**

Control Table Contents

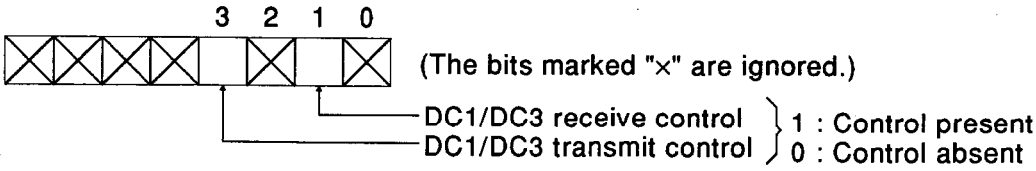
Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Signal Control, DC Control	The lower byte reads the signal control format, and the upper byte reads the DC control format.
[ ]%(2)	Dummy Data	This data has no meaning. (Used by the system as a work area.)
[ ]%(3)	Receive Buffer	Reads the size of the receive buffer.

- The communication control parameters of an open communication channel can be set or read out.
- The **procedure code** is 18 when setting control parameters, and 19 when reading them.
- The eight bits of the lower byte of [ ]%(1) are used to **set the signal control** as shown below:



*1 : For the RS-232C, CS control is performed regardless of the contents of the settings.

- The first and third bits of the eight bits of the upper byte of [ ]%(1) are used to set the **DC** control as shown below:



POINT

Since DC control and signal control must be set using both the lower and upper bytes, a hexadecimal number should be used as follows:

[ ]%(1)=&H [ ] [ ] [ ] [ ]

The lower byte is represented by two digits of the hexadecimal number.

The upper byte is represented by two digits of the hexadecimal number.

- The **receive buffer size** is set in byte units as follows:

0	: No receive buffer (Data received is ignored until commands to receive data are executed.)
1 to 1024 bytes	: Receive buffer exists (It is recommended that 1024 bytes be set.)

- [ ]%(2) is used by the BASIC system, and its value has no meaning.

POINT

When the control parameters are set, the receive buffer is completely cleared, and the condition of the control signals are all initialized.

PROGRAMMING EXAMPLE

```
1    'Set and read the communication control parameters of a communication channel
10   DIM TBL1%(2),TBL2%(3),TBL3%(3),B%(1)      :'Define arrays
20   CH%1                                      :'Define the channel number
30   TBL1%(0)=4800                             :'Set baud rate
40   TBL1%(1)=&H8                              :'Set character length and parity bit
50   TBL1%(2)=&H2                              :'Set stop bit
60   ZOPEN #CH%,TBL1%( )                       :'Open the communication channel
70   TBL2%(0)=18                               :'Specify "set communication control
                                                parameters"
80   TBL2%(1)=&HA03                            :'Set DC1/DC3 transmit control
90   TBL2%(3)=1024                             :'Set the receive buffer size to 1024 bytes
100   ZCNTL #CH%,0,TBL2%( )                    :'Execute
110   TBL3%(0)=19                              :'Specify "read communication control
                                                parameters"
120   ZCNTL #CH%,0,TBL3%( )                    :'Execute
130   PRINT The communication control parameters setting status:'Display the results that
                                                were read out
140   A$=RIGHT$("0000"+HEX$(TBL3%(1)),4)
150   L$=RIGHT$(A$,2)
160   H$=LEFT$(A$,2)
170   B%(0)=VAL("&H"+L$)                       :'The lower byte of TBL%(3)
180   B%(1)=VAL("&H"+H$)                       :'The upper byte of TBL%(3)
190   B=RDSET(1,B%(0))                         :'Find the first bit of the lower byte
200   IF B=1 THEN PRINT "wiht DC1\DC3receive control"
210   IF B<>1 THEN PRINT "without DC1\DC3receive control"
220   C=RDSET(3,B%(0))                         :'Find the third bit of the lower byte
230   IF C=1 THEN PRINT "with DC1\DC3send control"
240   IF C<>1 THEN PRINT "without DC1\DC3send control"
250   FOR I=1 TO 4
260    D=RDSET(I-1,B%(1))                      :'Find the first three bits of the upper
                                                byte
270    ON I GOSUB 320,350,380,410
280    D=0
290   NEXT I
300   ZCLOSE
310   END
320   IF D=1 THEN PRINT "with RScontrol"
330   IF D<>1 THEN PRINT "without RScontrol"
340   RETURN
350   IF D=1 THEN PRINT "with ERcontrol"
360   IF D<>1 THEN PRINT "without ERcontrol"
370   RETURN
380   IF D=1 THEN PRINT "with CScontrol"
390   IF D<>1 THEN PRINT "without CScontrol"
400   RETURN
410   IF D=1 THEN PRINT "with DRcontrol"
420   IF D<>1 THEN PRINT "without DRcontrol"
430   RETURN
```

Procedure Code 22 **Set the break characters**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Character Count	Specifies the number of break characters.
[ ]%(2)	Break Characters 1 and 2	The lower byte specifies the code of break character 1 and the upper byte specifies the code of break character 2.
[ ]%(3)	Break Characters 3 and 4	The lower byte specifies the code of break character 3 and the upper byte specifies the code of break character 4.
[ ]%(4)	Break Character 5	The lower byte specifies the code of break character 5. The upper byte is ignored.

Procedure Code 23 **Read out the break characters**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Character Count	Reads the number of break characters.
[ ]%(2)	Break Characters 1 and 2	The lower byte reads the code of break character 1 and the upper byte reads the code of break character 2.
[ ]%(3)	Break Characters 3 and 4	The lower byte reads the code of break character 3 and the upper byte reads the code of break character 4.
[ ]%(4)	Break Character 5	The lower byte reads the code of break character 5. The upper byte is ignored.

- The break characters of an open communication channel can be set or read out.
- The **procedure code** is 22 when setting break characters, and 23 when reading them.
- **Break Character Count** sets the number of characters that can terminate data reception.
  0 ..... No break characters.
  1 to 5 ..... Break characters are used.
- **Break Characters 1 to 5** set the actual break characters. Only the number of characters specified with break character count are set, all others are ignored.
  The codes that can be set for a break character are in the range 00_H FF_H.

After the data that was set as a break character is detected during the ZRECEIVE command execution, the data to receive is stored in the input element of command to receive data and tha execution of command to receive data is completed.

POINT

Since break characters 1 to 5 must be set using both the lower and upper bytes, a hexadecimal number should be used as follows:

[ ]%(2 to 4)=&H [ ] [ ] [ ] [ ]

The lower byte is represented by two digits of the hexadecimal number.

The upper byte is represented by two digits of the hexadecimal number.

PROGRAMMING EXAMPLE

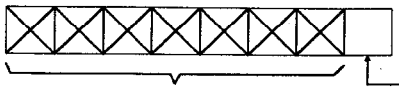
```
10   'Set and read break characters
20   DIM TBL1%(2),TBL2%(4),TBL3%(4)          :'Define arrays
30   A%=1                                    :'Communication channel CH1 (RS-232C)
40   TBL1%(0)=9600                           :'Set baud rate
50   TBL1%(1)=&H107                          :'Set character length and parity bit
60   TBL1%(2)=&H1                            :'Set stop bit
70   ZOPEN #A%,TBL1%( )                      :'Open the communication channel
80   TBL2%(0)=22                             :'Specify "set break characters"
90   TBL2%(1)=2                              :'Set the number of break characters
100   TBL2%(2)=&H30D                         :'Define the break characters
110   ZCNTL #A%,TBL2%( )                     :'Execute setting break characters
120   TBL3%(0)=23                            :'Specify "read break characters"
130   ZCNTL #A%,TBL3%( )                     :'Execute read break characters
140   PRINT "break characters length =";TBL3%(1):'Display
150   PRINT "break characters =&H";HEX$(TBL3%(2))
160   ZCLOSE #A%                             :'Close communication channel
170   END
```

Procedure Code 32 **Set the RS and ER signals ON/OFF**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	RS, ER Signals	The lower byte sets the RS signal ON/OFF, and the upper byte sets the ER signal ON/OFF.

- The RS and ER signals of an open communication channel can be forced ON or OFF. However, ER signal control cannot be performed with an RS-422 interface.
- The **procedure code** is 32.
- The eight bits of the upper and lower bytes of [ ]%(1) are used to set each signal OFF or ON as shown below:



(These bits are ignored.) signal control { 1 : ON / 0 : OFF

POINT

Since signals RS and ER must be set ON or OFF using both the lower and upper bytes, a hexadecimal number should be used as follows:

[ ]%(1)=&H [ ] [ ] [ ] [ ]

The lower byte is represented by two digits of the hexadecimal number.

The upper byte is represented by two digits of the hexadecimal number.

POINT

The RS/ER signals cannot be set ON/OFF for communication channels whose control parameter specifies signal control on.

PROGRAMMING EXAMPLE

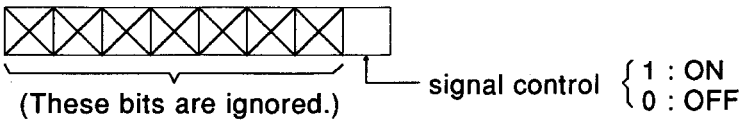
```
10  'Set the RS/ER signals ON/OFF
20  DIM TBL1%(2),TBL2%(3),TBL3%(1)          :'Define arrays
30  A%=1                                    :'Define the channel number
40  TBL1%(0)=9600                           :'Set baud rate
50  TBL1%(1)=&H108                          :'Set character length and parity bit
60  TBL1%(2)=&H1                            :'Set stop bit
70  ZOPEN #A%,TBL1%( )                      :'Open the communication channel
80  TBL2%(0)=18                             :'Specify "set communication control
                                             parameters"
90  TBL2%(1)=&H0                            :'Specify no RS/ER signal control
100  TBL2%(3)=1024                          :'Set the receive buffer size to 1024 bytes
110  ZCNTL #A%,0,TBL2%( )                   :'Execute
120  TBL3%(0)=32                            :'Specify "set RS/ER signals ON/OFF"
130  TBL3%(1)=&H101                         :'Set RS/ER signals ON
140  ZCNTL #A%,0,TBL3%( )                   :'Execute
150  ZCLOSE A#%
160  END
```

Procedure Code 33 **Read the ON/OFF status of the CS, DR, RS, ER and CD signals.**

Control Table Contents

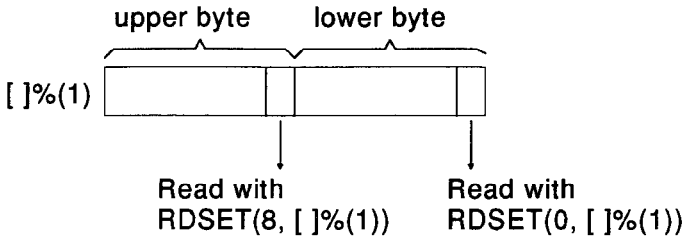
Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	CS, DR Signals	The lower byte reads the CS signal's ON/OFF status, and the upper byte reads the DR signal's ON/OFF status.
[ ]%(2)	RS, ER Signals	The lower byte reads the RS signal's ON/OFF status, and the upper byte reads the ER signal's ON/OFF status.
[ ]%(3)	CD Signal	The lower byte reads the CD signal's ON/OFF status, and the upper byte is ignored.

- The ON/OFF status of the CS, DR, RS, ER and CD signals of an open communication channel can be read.
- The **procedure code** is 33.
- The eight bits of the upper and lower bytes of each control table element are used to read each signal's ON/OFF status as shown below:



POINT

Since the condition of each signal is indicated by only the lowest bit of the lower and upper bytes, the RDSET command can be used to read the bit.



PROGRAMMING EXAMPLE

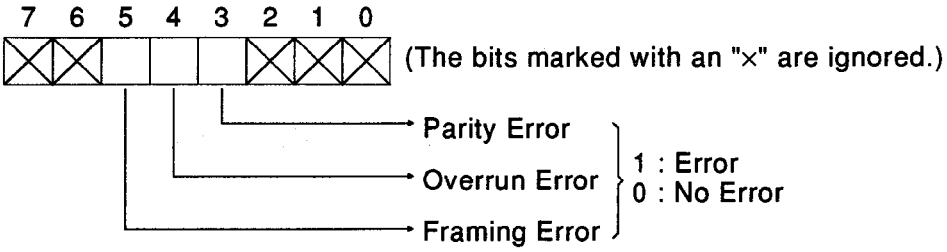
```
10  'Read the ON/OFF condition of the CS, DR, RS, ER and CD signals
20  DIM TBL1%(2),TBL2%(3)                    :'Define arrays
30  CH%=1                                    :'Set the communication channel
40  TBL1%(0)=9600
50  TBL1%(1)=&H108
60  TBL1%(2)=&H1
70  ZOPEN #CH%,TBL1%( )                      :'Open the communication channel
80  TBL2%(0)=33                              :'Specify "read conditions"
90  ZCNTL; #CH%,0,TBL2%( )                   :'Execute
100  A=RDSET(0,TBL2%(1))                     :'Read each bit
110  B=RDSET(8,TBL2%(1))
120  C=RDSET(0,TBL2%(2))
130  D=RDSET(8,TBL2%(2))
140  E=RDSET(0,TBL2%(3))
150  PRINT "CS signals=";                    :'Display conditions
160  IF A=1 THEN PRINT "ON "ELSE PRINT "OFF"
170  PRINT "DR signals=";
180  IF B=1 THEN PRINT "ON "ELSE PRINT "OFF"
190  PRINT "RS signals=";
200  IF C=1 THEN PRINT "ON "ELSE PRINT "OFF"
210  PRINT "ER signals=";
220  IF D=1 THEN PRINT "ON "ELSE PRINT "OFF"
230  PRINT "CD signals=";
240  IF E=1 THEN PRINT "ON "ELSE PRINT "OFF"
250  ZCLOSE #CH%                             :'Close the communication channel
260  END
```

Procedure Code 64 **Read the cause of a receive error.**

Control Table Contents

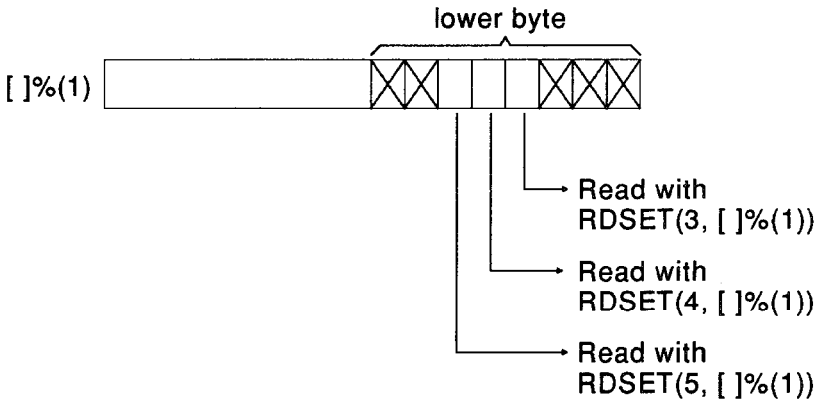
Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Error Cause	The lower byte reads the cause of the receive error.

- The cause of a receive error (parity, overrun or framing error) that occurred on an open communication channel can be read.
- The **procedure code** is 64.
- The eight bits of the lower byte of [ ]%(1) are used to read the **cause of the error** as shown below:



POINT

Since the cause of the error is indicated by bits 3, 4 and 5 of [ ]%(1), the RDSET command can be used to read each bit.



PROGRAMMING EXAMPLE

```
5    'Read the cause of the receive error
10   DIM TBL1%(2),TBL2%(1)                    :'Define arrays
20   CH%=1                                    :'Set the communication channel
30   TBL1%(0)=9600
40   TBL1%(1)=&H108
50   TBL1%(2)=&H1
60   ZOPEN #CH%,TBL1%( )                      :'Open the communication channel
70   TBL2%(0)=64                              :'Specify "read receive error cause"
80   ZCNTL #CH%,0,TBL1%( )                    :'Execute
90   A=RDSET(3,TBL2%(1))                      :'Read each bit
100   B=RDSET(4,TBL2%(1))
110   C=RDSET(5,TBL2%(1))
120   IF A=1 THEN PRITN "parity error!"       :'Display
130   IF B=1 THEN PRITN "overrun error!"
140   IF C=1 THEN PRITN "framing error!"
150   IF A<>1 AND B<>1 AND C<>1 THEN PRINT "no error"
160   ZCLOSE #CH%                              :'Close the communication channel
170   END
```

Procedure code 80 **Read the size of a receive buffer and the number of stored characters**

Control Table Contents:

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies a procedure code.
[ ]%(1)	Receive Buffer	Reads the size of a receive buffer.
[ ]%(2)	Number of Characters	Reads the number of characters stored in a receive buffer.

- The size of a receive buffer and the number of stored characters can be read.
- The procedure code is 80.
- Reading can be done for CH1(RS-232C), CH2(RS-232C) and CH3(RS-422).

Programming example

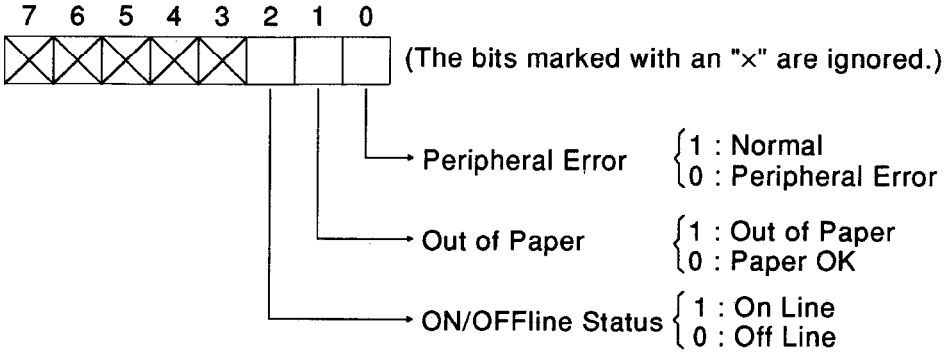
```
5'To read the size of a receive buffer and the number of stored characters:
10  DIM TBL1%(10), TBL2%(10)                    :'Define an array.
20  CH%=1                                       :'Specify the communication channel.
30  TBL1%(0)=9600
40  TBL1%(1)=&H108
50  TBL1%(2)=&H1
60  ZOPEN #CH%, TBL1%( )                        :'Open the communication channel.
70  TBL2%(0)=80                                 :'Read the size and number of characters.
80  ZCNTL #CH%, 0, TBL2%( )                     :'Execute
90  PRINT "Size of the receive buffer=" ;TBL2%(1)
                                                :'Display
100  PRINT "Number of characters      =";TBL2%(2)
110  ZCLOSE #CH%                                :'Close the communication channel.
120  END
```

Procedure Code 128 **Read the printer status.**

Control Table Contents

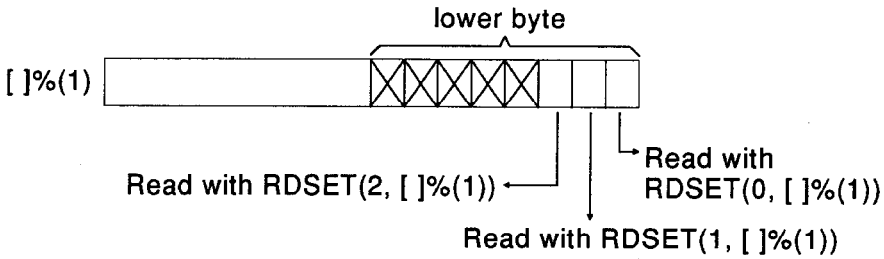
Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.
[ ]%(1)	Printer Status	The lower byte reads the printer status.

- The status (peripheral error, out of paper, ON/OFFline status) of a printer connected to an open parallel interface can be read.
- The **procedure code** is 128.
- The eight bits of the lower byte of [ ]%(1) are used to read the **printer status** as shown below:



POINT

Since the printer status is indicated by bits 0, 1 and 2 of [ ]%(1), the RDSET command can be used to read each bit.



The printer status reads the following parallel interface signals.

Peripheral Error ..... $\overline{\text{ERROR}}$
Out of Paper ..... PE
ON/OFFline ..... SLCT

The ON/OFF control for each signal depends on the printer, so please refer to the manual of the printer in use.

PROGRAMMING ERROR

```
10  'Read the printer status
20  DIM TBL%(1)                                  :'Define arrays
30  ZOPEN #4                                     :'Open the communication channel
40  TBL%(0)=128                                  :'Specify "read printer status"
50  ZCNTL #4,0,TBL%()                            :'Execute
60  A=RDSET(0,TBL%(1))                           :'Read each bit
70  B=RDSET(1,TBL%(1))
80  C=RDSET(2,TBL%(1))
90  PRINT "Printer"                              :'Display
100  IF A=1 THEN PRINT "Peripheral Normal" ELSE PRINT "Peripheral Error !!"
110  IF B=1 THEN PRINT "Out of Paper!!" ELSE PRINT "Paper OK"
120  IF C=1 THEN PRINT "On Line Status" ELSE PRINT "Off Line Status!!"
130  ZCLOSE #4
```

Procedure Code 136 **Send the initialize signal to the printer.**

Control Table Contents

Element Position	Item	Description
[ ]%(0)	Procedure Code	Specifies the procedure code.

- The initialize signal is sent for 100 msec to a printer connected to an open parallel interface.
- The **procedure code** is 136.
- The **initialize signal** is sent through the parallel interface $\overline{\text{INIT}}$ signal. The type of initialization performed by the printer according to this signal depends on the printer. Please refer to the manual for the printer in use.

PROGRAMMING EXAMPLE

```
10  'Send the initialize signal to the printer
20  DIM TBL%(0)                          :'Define an array
30  ZOPEN #4                             :'Open the communication channel
40  TBL%(0)=136                          :'Specify "send initialize signal to
                                          printer"
50  ZCNTL #4,0,TBL%()                    :'Execute
60  PRINT "Printer Initialize End"       :'Display "complete"
70  ZCLOSE #4                            :'Close the communication channel
80  END
```

- Enables or disables event signalling.

Z-EVENT

SYNTAX	**ZEVENTΔENABLEΔ<event number>** **ZEVENTΔDISABLEΔ<event number>**	
	ENABLE	Enables release (signalling) of the specified event.
	DISABLE	Disables release (signalling) of the specified event.
	event number	Specifies the number of the event to be enabled or disabled in the range 0 to 63.
EXAMPLE	**ZEVENT ENABLE 1**	Release of event number 1 is enabled.
	ZEVENT DISABLE 0	Release of event number 0 is disabled.

DESCRIPTION

- The **ZEVENT** command enables or disables release of events.
- The following occurs when event release is enabled:
  If the device designated for the event corresponding to **<event number>** when the event was defined is switched from OFF (0) to ON (1), or if a **ZSIGNAL** command is executed with the same event number, the event is signalled. If there are any programs waiting for release of the event, those programs become able to restart execution.
- The following occurs when event release is disabled:
  Even if the device designated for the event corresponding to **<event number>** when the event was defined is switched from OFF (0) to ON (1), the event is not signalled. Also, even if a **ZSIGNAL** command is executed with the same event number, the event is not signalled. Therefore, even of the above-mentioned device changes or if a **ZSIGNAL** command is executed, the OS will not restart execution.
- **<Event number>** must be previously defined with the **DEF ZEVENT** command.
- An event can be enabled or disabled several times. The most recent setting is valid.
- When an event is defined, its release is disabled.

REMARK

(1) For details on event control, see section 8.2.

(2) See DEF ZEVENT, ZSIGNAL and ZWAIT EVENT.

- Specifies the input source from which input will be received for the INPUT and similar commands.

Z Input DeVice

SYNTAX	**ZIDVΔ<channel number>**	
	channel number	Specifies the channel number of a communication port to which a console or terminal is connected.
EXAMPLE	**ZIDV 1**	Input for the INPUT and similar commands will be received from the terminal connected to CH1 (RS-232C).

DESCRIPTION

- The **ZIDV** command designates the input source to be used by the **INPUT** and similar commands.
- The input source designated by the **ZIDV** command is only valid for the following commands and functions:

  INPUT INKEY$
  INPUT$ LINE INPUT

- **<Channel number>** is used to select an input device connected to a communication port. Channel numbers and the corresponding ports are shown below:

  channel number communication port
  0..... Console designated by the AD51H system dip switch
  1..... CH1 (RS-232C)
  2..... CH2 (RS-232C)
  3..... CH3 (RS-422)

POINT

The communication channel to be designated by the ZIDV command should first be opened with the ZOPEN command. However, if zero is specified for the ZIDV command, it is not necessary to open the channel.
If the channel designated as the input source with the ZIDV command is closed with the ZCLOSE command, the input source becomes the console selected by the AD51H system dip switch (channel 0).

REMARK

See ZLDV, ZODV and ZOPEN. Also see section 7.3.

- Changes the printer port.

Z Line printer DeVice

SYNTAX	**ZLDVΔ<channel number>**	
	channel number	Specifies the channel number of the communication port to which the printer is connected.
EXAMPLE	**ZLDV 4**	Data from commands that send output to the printer will be sent to the printer connected to the parallel interface.

DESCRIPTION

- The **ZLDV** command designates the printer to be used by the **LLIST**, **LPRINT** and **LFILES** commands.
- **<Channel number>** is used to select the communication port to which the printer is connected. Channel numbers and the corresponding ports are shown below:

channel number	communication port
0	Printer connected to the PC/AT.
1	CH1 (RS-232C)
2	CH2 (RS-232C)
3	CH3 (RS-422)
4	CH4 (PARALLEL)

POINT

The communication channel to be designated by the ZLDV command should first be opened with the ZOPEN command. However, if zero is specified for the ZLDV command, it is not necessary to open the channel.
If the channel designated as the output source with the ZLDV command is closed with the ZCLOSE command, the output source becomes the printer set by the MS-DOS (channel 0).

REMARK

See ZIDV, ZODV and ZOPEN. Also see section 7.3.

- Defines a message port.

Z MESSAGE

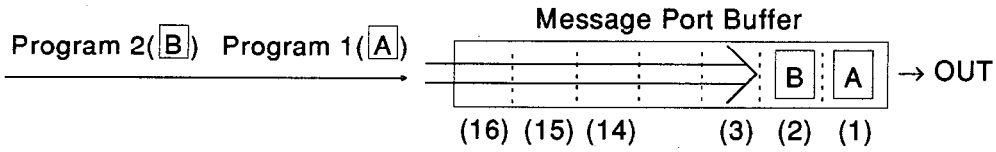
SYNTAX	**ZMESSAGEΔ<message port number>ΔLEN=<message length>**	
	message port number	Specifies the number of the message port to be defined.
	message length	Specifies the length of the messages to be used by the message port in byte units from 1 to 256.
EXAMPLE	**ZMESSAGE 1 LEN=128**	The message port with message port number 1 is defined with a message length of 128 bytes.

DESCRIPTION

- The **ZMESSAGE** command defines a message port in main memory to be used by programs running under multitasking to communicate with each other.
- Up to 32 message ports can be defined. To specify which message port is to be defined,**<message port number>** is used.
  The type of the message port depends on the **<message port number>**, as shown below.
  Please choose a type to suit your purpose.

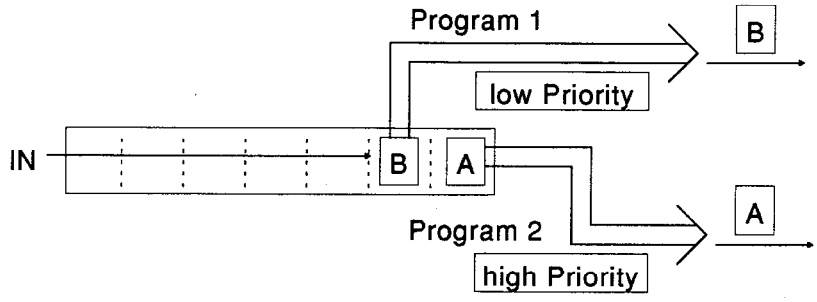
FIFO format ... Message port numbers 16 to 31

Data that is sent to the message port is stored in the order it was received.



PRIORITY format ... Message port numbers 0 to 15

The order in which data was received is irrelevant, data sent from the program with highest priority is retrieved first.



- **<Message length>** specifies the maximum length in byte units for messages that are sent to the message port. The range of valid settings is 0 to 256 bytes.
  The necessary byte values are shown below. If the **<output device>** is specified by **ZMESSAGE PUT** then the total size of the output device will be determined by these necessary byte values.

  integer data ..... 1 data item = 2bytes
  single precision real number data ..... 1 data item = 4 bytes
  double precision real number data ..... 1 data item = 8 bytes
  character data ..... 1 character = 1 byte

  It is suggested that 256 bytes be specified for the message length.

POINT

Each message port should be defined with the ZMESSAGE command in only one of the programs that are executing simultaneously under multitasking.

- In order to use the message port, after defining it with the **ZMESSAGE** command, it must also be opened with the **ZMESSAGE OPEN** command.

PROGRAMMING EXAMPLE

```
10  'Exchange data between tasks using the message port (TASK 1)
20  DEF Z EVENT1                          :'Define event 1
30  ZEVENT ENABLE 1                       :'Enable release of event 1
40  A$=SPACE$(80)                         :'Store dummy data to a character variable
50  ZMESSAGE 1 LEN=80                     :'Define message port 1
60  ZMESSAGE OPEN 1                       :'Open message port 1
70  ZSIGNAL 1                             :'Signal event 1
80  ZMESSAGE PUT 1,"MESSAGE FOR TASK2"    :'Write the message
90  ZMESSAGE CLOSE 1                      :'Close message port 1
100  ZMESSAGE  2 LEN=80                   :'Define message port 2
110  ZMESSAGE OPEN 2                      :'Open message port 2
120  ZMESSAGE GET 2,A$                    :'Read the message
130  ZMESSAGE CLOSE 2                     :'Close message port 2
140  PRINT A$                             :'Display
150  ZMESSAGE KILL 1                      :'Delete message port 1
160  ZMESSAGE KILL 2                      :'Delete message port 2
170  END
```

```
10  'Exchange data between tasks using the message port (TASK 2)
20  ZWAIT EVENT 1                         :'Wait for event 1
30  B$=SPACE$(80)                         :'Store dummy data to a character variable
40  ZMESSAGE OPEN 1                       :'Open message port 1
50  ZMESSAGE GET 1,B$                     :'Read the message
60  ZMESSAGE CLOSE 1                      :'Close message port 1
70  ZMESSAGE OPEN 2                       :'Open message port 2
80  ZMESSAGE PUT 2,"MESSAAGE FOR TASK1"   :'Write the message
90  ZMESSAGE CLOSE 2                      :'Close message port 2
100  LOCATE 0,1                           :'Change display position
110  PRINT B$                             :'Display
120  END
```

REMARK

(1) For information on program priority, see sections 8.2, and 8.4 the ZURGENCY command.

(2) See ZMESSAGE CLOSE, ZMESSAGE GET, ZMESSAGE KILL, ZMESSAGE OPEN and ZMESSAGE PUT.Also see section 8.5.2.

- Closes a message port.

ZMESSAGE CLOSE

SYNTAX	**ZMESSAGEΔCLOSEΔ[<message port number>]**

	<message port number>	Specifies the number of the message port to be closed.
EXAMPLE	**ZMESSAGE CLOSE 1**	The message port with message port number 1 is closed.
	ZMESSAGE CLOSE	All open message ports are closed.

DESCRIPTION

- The **ZMESSAGE CLOSE** command closes open message ports.
- **<Message port number>** specifies the number of the message port to be closed.
  If this is omitted, all open message ports are closed.
- The program that executed the **ZMESSAGE CLOSE** command can no longer use the specified message port.
- A message port that was closed with the **ZMESSAGE CLOSE** command can be reopened with the **ZMESSAGE OPEN** command.
- When deleting a message port, be sure that the port has been closed with the ZMESSAGE CLOSE command by all programs executing under multitasking before executing the **ZMESSAGE KILL** command.

REMARK

See ZMESSAGE, ZMESSAGE GET, ZMESSAGE KILL, ZMESSAGE OPEN and ZMESSAGE PUT. Also see section 8.5.2.

- Reads a message from the message port.

ZMESSAGE GET

SYNTAX	**ZMESSAGEΔGETΔ<massage port number>,<variable name>, . . [TIMEOUTΔ<timeout period>].**

<message port number> Specifies the number of the message port from which to read data.

<variable name> Specifies the variable in which to store the data that is read.

<timeout period> Specifies the maximum amount of time to wait for a message to be written to the message port.

The time is specified as "HH:MM:SS:R"

HH	: hours	(0 to 23)
MM	: minutes	(0 to 59)
SS	: seconds	(0 to 59)
R	: 100 ms	(0 to 9 indicates 0 to 900 milliseconds)

EXAMPLE

ZMESSAGE GET 1,AS

Character data is read from message port number 1 and stored to A$.

ZMESSAGE GET 16,A%TIMEOUT "0:1:0:0"

Numeric data is read from message port number 16 and stored to A%. If after one minute a message has not been written, a timeout error occurs.

DESCRIPTION

- The **ZMESSAGE GET** reads a message that was written to a message port. If no messages are written in the port, program execution is paused until a message is written.
- **<Message port number>** specifies the number of the message port from which the message will be read.
- **<Variable name>** specifies the variable to which the data that is read from the message port will be stored. The type of data in the message port must match the type of variable specified for **<variable name>**.

POINTS

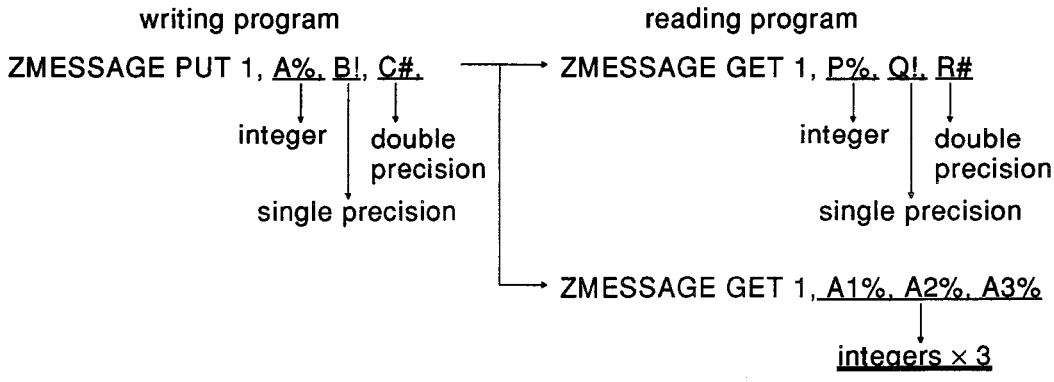
(1) Dummy data equal to or longer than the message length must be stored to the variable specified for <variable name> before the ZMESSAGE GET command is executed. If dummy data is not stored, when the ZMESSAGE GET command is executed, an error occurs.

numeric variables [ ]% = 0 [ ]! = 0 [ ]# = 0
character variables [ ]S = SPACE$(255)

(2) If an array variable is specified for variable name, even if fewer than ten elements are used, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the ZMESSAGE GET command is executed, an error occurs.
(Normally, if less than ten elements of an array are used, it is not necessary to define the array with the DIM command.)

- Multiple variables can be specified for **<variable name>** if all are numeric value variables. The variables must be separated by commas, and must correspond correctly with the type of data written with the **ZMESSAGE PUT** command. If the variables do not correspond correctly, the messages will be read incorrectly.

Example



- **<Timeout period>** specifies the maximum amount of time the program will wait for a message to be written to the message port, if one is not already written. It is specified as shown below: (while the program is waiting, execution is paused.)

  "HH:MM:SS:R"

  HH ..... hours ("0" to "23") MM ..... minutes ("0" to "59")

  SS ..... seconds ("0" to "59") R ..... 100 ms ("0" to "9" indicates 0 to 900 milliseconds)

  If period is not specified, or if it is set to "0:0:0:0", the program will wait indefinitely.

REMARKS

See ZMESSAGE, ZMESSAGE CLOSE, ZMESSAGE KILL, ZMESSAGE OPEN and ZMESSAGE PUT. Also see section 8.5.2.

- Deletes a defined message port.

ZMESSAGE KILL

SYNTAX	**ZMESSAGEΔKILLΔ<message port number>**	
	<message port number>	Specifies the number of the message port to be deleted.
EXAMPLE	**MESSAGE KILL 1**	The message port with message port number 1 is deleted.

DESCRIPTION

- The **ZMESSAGE KILL** command deletes message ports that have been defined in main memory.
- **<Message port number>** specifies the number of the message port to be deleted.
- When deleting a message port, be sure that the port has been closed with the **ZMESSAGE CLOSE** command by all programs executing under multitasking before executing the **ZMESSAGE KILL** command.
  If a program exists in which the message port is not closed, when the **ZMESSAGE KILL** command is executed, an error occurs.

POINT

Each message port should be deleted with the ZMESSAGE KILL command in only one of the programs that are executing simultaneously under multitasking.

REMARK

See ZMESSAGE, ZMESSAGE CLOSE, ZMESSAGE GET, ZMESSAGE OPEN and ZMESSAGE PUT. Also see section 8.5.2.

- Opens a defined message port.

ZMESSAGE OPEN

SYNTAX	**ZMESSAGEΔOPENΔ<message port number>**	
	<message port number>	Specifies the number of the message port to be opened.
EXAMPLE	**ZMESSAGE OPEN 1**	The message port with message port number 1 is opened.

DESCRIPTION

- The **ZMESSAGE OPEN** command opens message ports that have been defined with the **ZMESSAGE** command.
- **<Message port number>** specifies the number of the message port to be opened.
- The **ZMESSAGE** command only defines a message port. The port must then be opened with the **ZMESSAGE OPEN** command in order to be used.

POINTS

(1) If the ZMESSAGE OPEN command is executed on a message port that has not been defined with the ZMESSAGE command, an error occurs.

(2) Each message port should be opened with the ZMESSAGE OPEN command in all of the programs that are executing simultaneously under multitasking.

REMARK

See ZMESSAGE, ZMESSAGE CLOSE, ZMESSAGE GET, ZMESSAGE KILL and ZMESSAGE PUT. Also see section 8.5.2.

- Writes the next message to the message port.

ZMESSAGE PUT

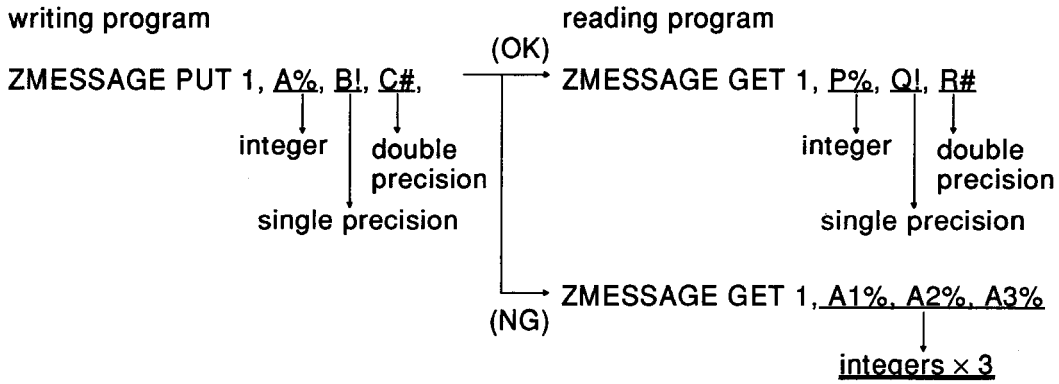
SYNTAX	**ZMESSAGEΔPUTΔ<message port number>,<message>, . . .**	
	<message port number>	Specifies the number of the message port to which to write the message.
	<message>	Specifies the message to be written to the message port.
EXAMPLE	**ZMESSAGE PUT 1,"Hello"**	The message "Hello" is written to message port number 1.
	ZMESSAGE PUT 16,A%	The contents of A% are written to message port number 16.

DESCRiPTION

- The **ZMESSAGE PUT** command writes a message to a message port.
- **<Message port number>** specifies the number of the message port to which the message will be written.
- Numeric constants, character string constants, numeric variables, character string variables and array variables may be specified for **<message>**. Character strings must be enclosed in quotes.
- Multiple numeric values can be specified for **<message>** by separating them with commas.

  In this case, the multiple messages written with the ZMESSAGE PUT command must correspond correctly with the type of data read by the **ZMESSAGE GET** command. If the messages do not correspond correctly with the variables, the messages will be read incorrectly.

Example



REAMRK

See ZMESSAGE, ZMESSAGE CLOSE, ZMESSAGE GET, ZMESSAGE KILL and ZMESSAGE OPEN. Also see section 8.5.2.

- Transfers data between variables.

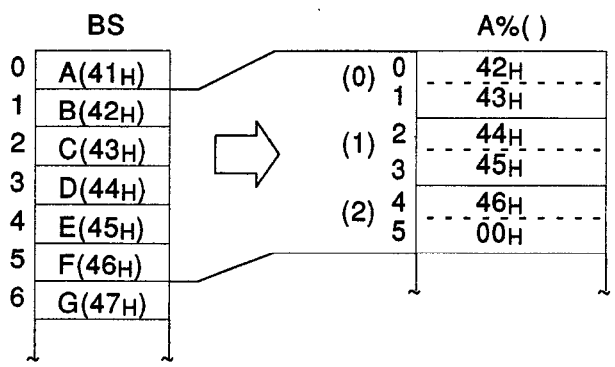
Z MOVE

SYNTAX	**ZMOVEΔTOΔ<transfer destination>,<offset 1>ΔFROM Δ<transfer source>, <offset 2>,<number of bytes>**

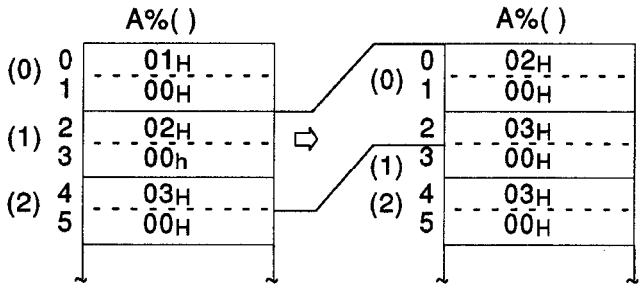
transfer destination	Specifies a variable or one-dimensional array to which the data will be moved.
offset 1	Specifies the storage position of the transfer destination data within the variable or array in byte units between 0 and 65534.
transfer source	Specifies a variable or one-dimensional array from which the data will be moved.
offset 2	Specifies the storage position of the transfer source data within the variable or array in byte units between 0 and 65534.
number of bytes	Specifies how many bytes of data to transfer.

EXAMPLE

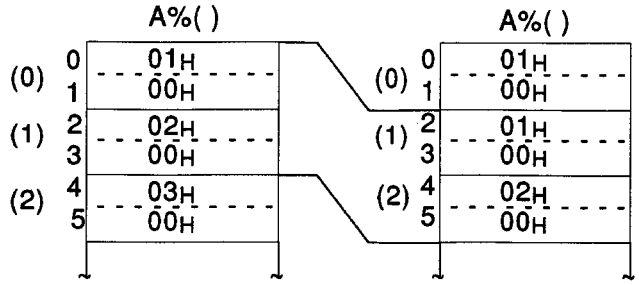
ZMOVE TO A%( ),0 FROM B$,1,6 ..... Six bytes starting with the first byte (2nd character) of B$ are transferred to A%(0) to A%(2).



ZMOVE TO A%( ),0 FROM A%( ),2,3 ..... The data in A%(1) and the lower byte of A%(2) is transferred to A%(0) and the lower byte of A%(1).

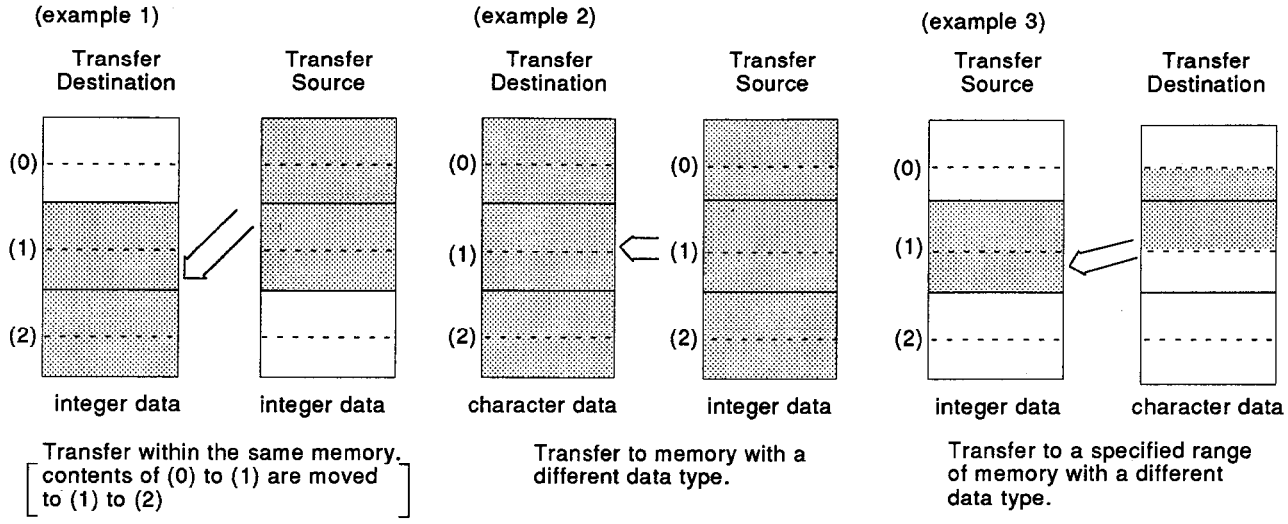


ZMOVE TO A%( ),2 FROM A%( ),0,4 ..... The data in A%(0) to A%(1) is transferred to A%(1) to A%(2).



DESCRIPTION

- The **ZMOVE** command moves data transparently and regardless of data type from the **<transfer source>** to the **<transfer destination>** in a BASIC program. The amount of data to transfer is specified with **<number of bytes>**.
- The head position of **<transfer source>** or **<transfer destination>** can be specified as an offset in byte units from the designated memory address.
- This command can perform data transfers that the **LET** command cannot (numeric value ↔ character string).



- The variable which is to have its contents changed to be identical with **<transfer source>** should be specified for **<transfer destination>**.

POINTS

(1) Dummy data should be stored to the variable that is specified for <transfer destination> before the ZMOVE command is executed. If dummy data is not stored, an error will occur when the ZMOVE command is executed.

integer variables	**[ ]% = 0**
real number variables	**[ ]! = 0, [ ]# = 0**
character variables	**[ ]$ = SPACE$(255)**

(2) If an array variable is specified for <transfer destination>, even if the number of elements used is less than 10, the array must be defined with the DIM command.
For character string array variables, dummy data should be stored to the elements to be specified as shown below:

[ ]$ (n) = SPACE$(255)

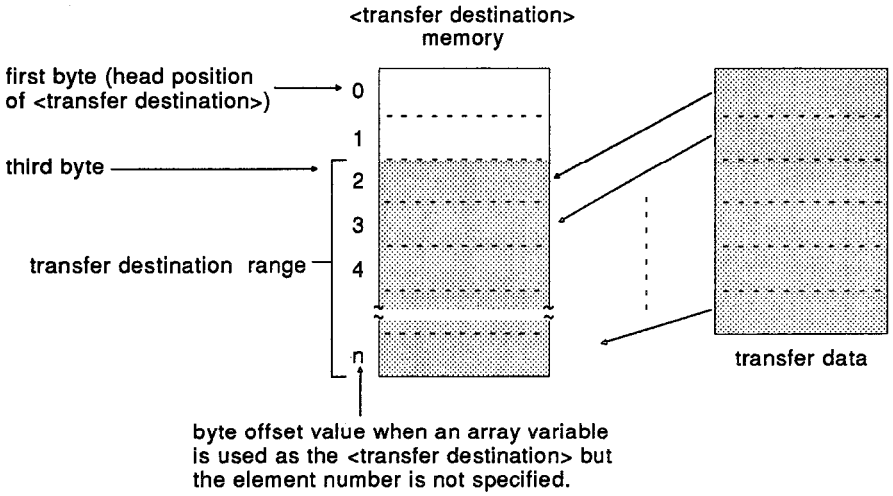
When the ZMOVE command is executed, only the data in the element specified becomes the destination storage range.

(3) Two dimensional array variables may not be used with the ZMOVE command.

- The **<offset 1>** designates the head position of the memory range for the transfer, and is specified in byte units as an offset from **<transfer destination>**.

Example

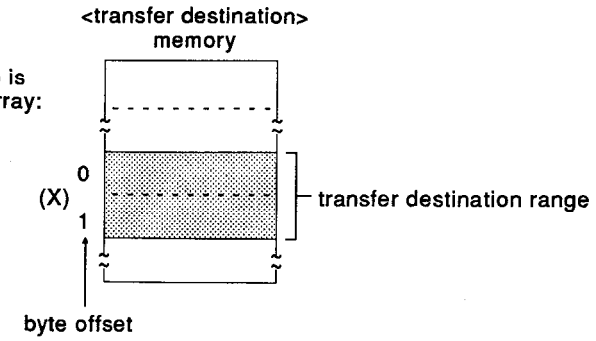
To store the transferred data starting at the third byte, **<offset 1>** to 2.



When using an array variable as the **<transfer destination>**, the byte offset should be set as an offset from the specified array element.

Example

If the element position (x) is specified for an integer array:



- The variable which contains the data to be transferred to **<transfer destination>** should be specified for **<transfer source>**.

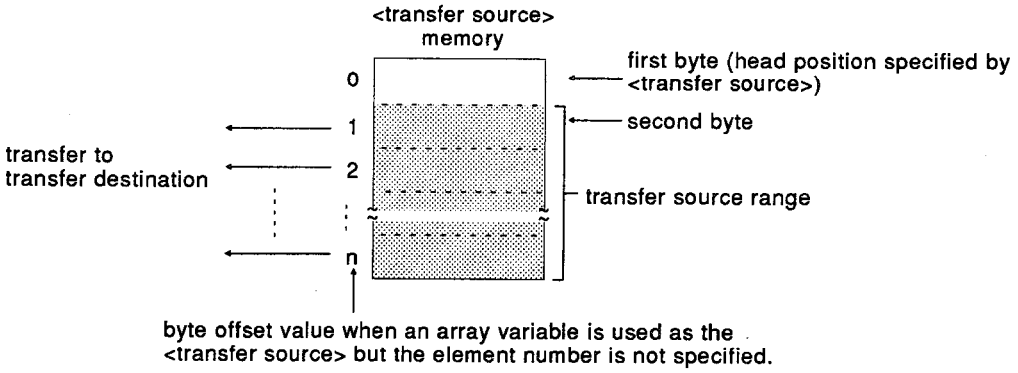
POINT

The array variable specified for <transfer source> must be defined with the DIM command even if the number of elements used is less than 10. When the ZMOVE command is executed, only the data from the part of the array defined with the DIM command is transferred. (Normally, arrays with fewer than ten elements do not need to be defined with the DIM command.)

- **<Offset 2>** designates the head position of the memory range for the transfer, and is specified in byte units as an offset from **<transfer source>**.

Example

To transfer data from the second byte, set **<offset 2>** to 1.



When using an array variable as the **<transfer source>**, the byte offset should be set as an offset from the specified array element.
The method for specifying an array variable as the **<transfer source>** is the same as for **<transfer destination>**. Please refer to the previous section.

- **<Number of bytes>** specifies the number of bytes of data to transfer from the head position specified with **<offset 2>** of **<transfer source>** to the head position specified with **<offset 1>** of **<transfer destination>**.
  If the number of bytes specified for **<transfer source>** is different from the number of bytes specified for **<transfer destination>**, the smaller value is assumed.

- The values that can be specified for **<offset 1>**, **<offset 2>**, **<number of bytes>** are described by the following conditional expressions. Both conditions must be satisfied.

  0≤ (offset 1+number of bytes) ≤number of bytes of destination memory
  0≤ (offset 2+number of bytes) ≤number of bytes of source memory

- If the number of bytes in the **<transfer destination>** range, the number of bytes in the **<transfer source>** range, and the **<numbre of bytes>** is different, the smallest value is assumed.
  Memory contents are normally completed after the transfer.
  The contents of any unused portion of the **<transfer destination>** memory remains as it was before the transfer (it is not changed).

- The representation of numeric constants used for **<offset 1>**, **<offset 2>** and **<number of bytes>** is decimal (base 10), but if a value above 32767 is specified, the representation is hexadecimal. 32768 to 65535 are represented as &H8000 to &HFFFF. If a decimal value in the range -32768 to -1 is specified, it is considered the same as if the corresponding number between 32768 to 65536 were specified.

PROGRAMMING EXAMPLE

```
10  'Transfer data from numeric value array variable A%() to character string variable B$
20  DIM A%(9)                                :'Define arrays
30  A%(0)=&H4241                             :'Store data
40  A%(1)=&H4443
50  PRINT "A%(0)=";HEX$(A%(0))               :'Display data before transfer
60  PRINT "A%(1)=";HEX$(A%(1))
70  PRINT
80  B$=SPACE$(255)                           :'Store dummy data
90  ZMOVE TO B$,0 FROM A%( ),0,4             :'Transfer data
100  PRINT "B$";B$                           :'Display data after transfer
110  END
```

- Changes the output destination used by the PRINT command and similar commands.

Z Output DeVice

SYNTAX	**ZODVΔ<channel number>**	
	channel number	Specifies the channel number of the communication port to which a console or terminal is connected.
EXAMPLE	**ZODV 1**	Data from commands that send output such as the PRINT command will be sent to the terminal connected to CH1 (RS-232C).

DESCRIPTION

- The **ZODV** command designates the output destination to which data from the **PRINT** and similar commands is sent.

POINT

The output destination for the printer is set with the ZLDV command.

- Only data from the following commands and functions is sent to the output destination specified with the **ZODV** command.

CLS	PRINT USING
LOCATE	SPC
PRINT	TAB

- **<Channel number>** is used to select the communication port to which the device to receive output data is connected. Channel numbers and the corresponding ports are shown below:

channel number	communication port
0	Console designated by the AD51H system dipswitch.
1	CH1 (RS-232C)
2	CH2 (RS-232C)
3	CH3 (RS-422)
4	CH4 (PARALLEL)

POINT

The communication channel to be designated by the ZODV command should first be opened with the ZOPEN command. However, if zero is specified for the ZODV command, it is not necessary to open the channel.
If the channel designated as the output destination with the ZODV command is closed with the ZCLOSE command, the output destination becomes the console selected by the AD51H system dipswitch (channel number 0).

REMARK

See ZIDV, ZLDV and ZOPEN. Also see section 7.3.

- Opens the channel number of a communication port in order to prepare for communication with an external device.

Z OPEN

SYNTAX	**ZOPENΔ[#]<channel number>[<,control table>]**

	channel number	Specifies the communication port which is to be used for communication with external devices.
	control table	Specifies the parameters needed for communication with external devices.
EXAMPLE	**ZOPEN #1,A%( )**	The AD51H CH1 (RS-232C) port is opened using the contents of integer array variable A%( ) as the control table.
	ZOPEN #4	The AD51H parallel interface is opened.

DESCRIPTION

- The **ZOPEN** command opens a communication port and enables communication with an external device.
- **<Channel number>** specifies which AD51H communication port will be used. The channel numbers correspond to the communication ports as shown below:

channel number	communication port
1	CH1 (RS-232C)
2	CH2 (RS-232C)
3	CH3 (RS-422)
4	CH4 (PARALLEL)

- The **<control table>** specifies various parameters necessary for communication. For channel number 4 (parallel), however, it is not necessary to set the **<control table>**. Therefore, it should be omitted.
- An integer array variable should be used for the **<control table>** to set the following data:

  [ ]%(0) .... The baud rate is specified.
  [ ]%(1) .... The parity bit is specified in the lower byte, and the character length is specified in the upper byte.
  [ ]%(2) .... The stop bit is specified.

POINT

Although the <control table uses only three array elements, the array variable must be defined with the DIM command. If the array is not defined with the DIM command, when the ZOPEN command is executed, an error occurs.
(Normally, arrays with fewer than ten elements do not need to be defined with the DIM command.)

Baud Rate
Select a value from the following list for the baud rate setting:

1200, 2400, 4800, 9600, 19200 BPS

Character Length
Select a value from the following list for the character length setting:

5, 6, 7, 8

Parity Bit
Select one of the following settings for the parity bit:

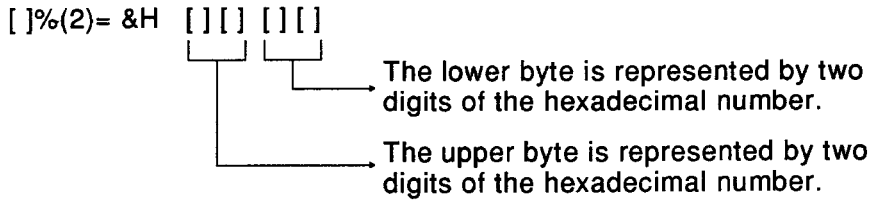
No Parity : 0, Even Parity : 1, Odd Parity : 2

Stop Bit
Select one of the following settings for the stop bit:

1 bit : 1, 2 bits : 2, 1.5 bits : 3

POINT

Since the character length and parity bit must be set using both the lower and upper bytes, a hexadecimal number should be used as follows:



- Before changing the input or output port with the **ZODV, ZIDV** or **ZLDV** commands, the port must be opened with the **ZOPEN** command.
- If the **ZOPEN** command is executed using the port designated as the console or debug port with the AD51H system dip switch, an error occurs.
- If **<channel numbers>** 1 to 3 are specified, a 1024 K communication buffer is reserved and its contents are cleared.
  To perform communication control or to set the size of the buffer, the **ZCNTL** command should be used.

REAMRK

See ZCLOSE. Also see chapter 7.

- Receives data from the communication port.

Z RECEIVE

SYNTAX	**ZRECEIVE△#<channel number>, 0, <control table>,<input element>**

	channel number	Specifies the communication port from which to receive data.
	control table	Specifies parameters for receiving data.
	input element	Specifies the variable or array in which to store the received data.
EXAMPLE	**ZRECEIVE #1,0,A%( ),DA$**	Data is received from CH1 (RS-232C) according to the parameters specified in integer array A%(0) and A%(2). The data is stored in DA$, and the number of bytes of data received is stored in A%(1).

DESCRIPTION

- The **ZRECEIVE** command receives data from the communication port specified with **<channel number>** according to the parameters set with **<control table>** and stores the data in **<input element>**.
- **<Channel number>** specifies which AD51H communication port to use. The channel numbers correspond to the communication ports as shown below:

  Channel Number Communication Port

  1 ..... CH1 (RS-232C)

  2 ..... CH2 (RS-232C)

  3 ..... CH3 (RS-422)

  Before executing the **ZRECEIVE** command to receive data from a communication port,the port must first be opened with the **ZOPEN** command. If the port is not opened,an error occurs.
- The **<control table>** is used to specify the number of bytes of data to receive,and the timeout period. Also, the actual number of bytes of data received is stored here.
  An integer array is used for the **<control table>**.

  [ ]%(0) ..... Specifies the requested number of bytes to receive.

  [ ]%(1) ..... The number of bytes received is stored following completion of the transfer.

  [ ]%(2) ..... Specifies the timeout period.

POINT

Although the <control table> uses only three array elements, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the ZRECEIVE command is executed, an error occurs. (Normally, if an array uses less than ten elements, it is not necessary to define it with the DIM command.)

(1) Requested number of bytes ..... [ ]%(0)

Specifies the number of bytes of data to receive. Since the method of counting bytes depends on the contents of **<input element>**, refer to the following table:
If an integer variable is used as the input element, 2 bytes are received.
If an integer array is used as the input element, 2 to 1024 bytes can be received.
If a character array is used as the input element, 1 to 256 bytes can be received.

(2) Timeout period ..... [ ]%(2)

Specifies the wait time for receiving data. When the **ZRECEIVE** command is executed, if reception of data is not completed within a specified time period,an error occurs. The timeout period is set to the specified value X 100ms.
If a timeout error occurs, the number of bytes of data received before the timeout error is stored to [ ]%(1), and all data received before the timeout error is stored to **<input element>**.
Values in the range 0 to 65535 may be specified, but when specifying values greater than 32767, hexadecimal representation should be used (decimal notation will cause an error when it is stored). If zero is specified, the timeout period is unlimited.

(3) Number of bytes received ..... [ ]%(1)

The number of bytes received is automatically stored following completion of the data reception. If data was received normally, this value is the same as "requested number of bytes", so the value is used to check the condition of the data transfer.

- **<Input element>** is the storage area for the data that is received. An integer variable, integer array, character string variable or character string array may be specified.

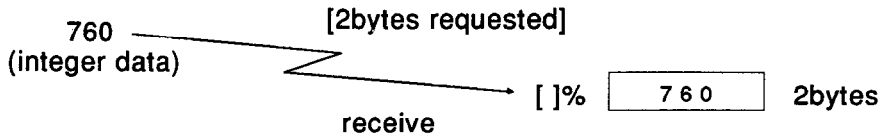
POINTS

(1) Before executing the ZRECEIVE command, dummy data must be stored to the variable to be used as the input element. The dummy data must be at least as long as the requested number of bytes to receive. If dummy data is not stored, when the ZRECEIVE command is executed, an error occurs.

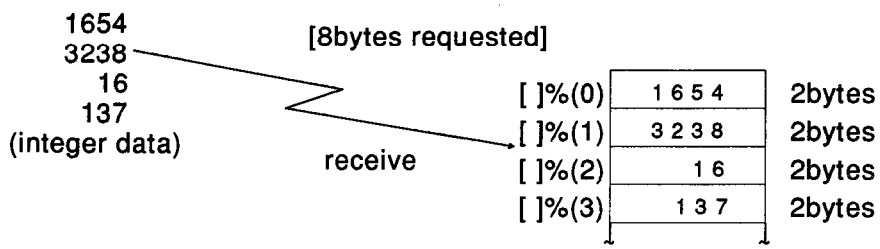
integer variables	**[ ]%= 0**
character variables	**[ ]$ = SPACE$ (255)**

(2) Even if the array specified as the input element uses less than ten elements, it must be defined with the DIM command. If it is not defined with the DIM command, when the ZRECEIVE command is executed, an error occurs.

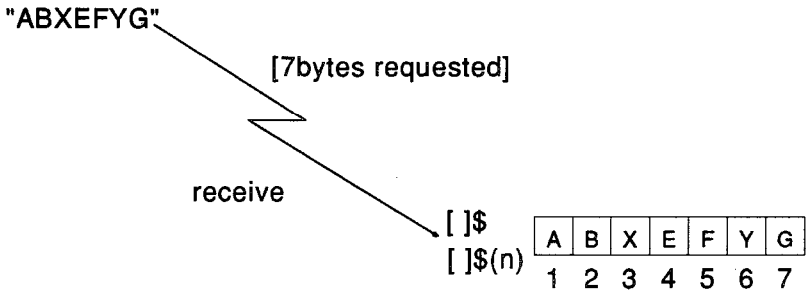
(1) When an **integer variable** is specified, one integer value can be received. Since one integer value is handled as two bytes, the "requested number of bytes" should be set to 2.



(2) When an **integer array variable name** is specified, the number of bytes of data specified with "requested number of bytes" can be received. Since one integer value is handled as two bytes, the "requested number of bytes" should be set as follows:
number of integer values to receive X 2



(3) When a **character string variable** or **character array** is specified, the number of bytes of data specified with "requested number of bytes" can be received. Since one character is handled as one byte, the "requested number of bytes" should be set the same as the number of data items to receive.



- A channel number that has not been opened by the **ZOPEN** command cannot be specified.
  If a closed channel is specified, an error occurs.

- If the requested number of bytes cannot be stored in the variable or array specified for the **<input element>**, an error occurs, and the command is not executed.

- If a character that was defined as a break character with the **ZCNTL** command is detected while data is being received, data transfer is aborted. If this occurs,the number of bytes of data received before the break character was detected is stored to [ ]%(1) in the control table, and all data received up until the break character was detected is stored in **<input element>**.

PROGRAMMING EXAMPLE (executed in AD51H BASIC)

```
10  'Receive data from the IBM PC/AT (AD51H receive side)
20  ON ERROR GOTO 360                          :'Branch to line 360 in case of error
30  '
40  '***** RS-232 CH.OPEN *****
50  DIM TBL1%(2)                               :'Define an array
60  CH%=1                                      :'Define a communication channel
70  TBL1%(0)=4800                              :'Set the baud rate
80  TBL1%(1)=&H8                               :'Set the character length and parity bit
90  TBL1%(2)=&H1                               :'Set the stop bit
100  ZOPEN #CH%,TBL1%( )                       :'Open the communication channel
110  '
120  '***** PORT CNTL *****
130  DIM CNT%(10)                              :'Define an array
140  CNT%(0)=32                                :'RS, ER, signal control W
150  CNT%(1)=&H1                               :'RS control NM
160  ZCNTL #CH%,CNT%( )                        :'Signal control S
170  '
180  '***** DATA RECEIVE *****
190  DIM TBL2%(2)                              :'Define an array
200  TBL2%(0)=12                               :'Specify the number of bytes to receive
210  TBL2%(1)=0                                :'Store the number of bytes received
220  TBL2%(2)=300                              :'Set the timeout period
230  WOR%=TBL2%(0)/2                           :'Read two bytes
240  DIM CTBL%(WOR%-1)                         :'Define an array
250  ZRECEIVE #CH%,0,TBL2%( ),CTBL%( )         :'Receive data
260  PRINT "RECEIVE CHARACTER=";TBL2%(1)       :'Display the number of characters received
270  FOR I%=0 TO WOR%-1
280  PRINT "RECEIVE DATA =&H";HEX$(CTBL%(I%)):'Display the data received
290  NEXT I%
300  '
310  '***** RS-232 CH.CLOSE *****
320  ZCLOSE #CH%                               :'Close the communication channel
330  END                                       :'Finish execution
340  '
350  '***** ERROR ROUTINE *****
360  ERTYPE%=ERR:ERLINE%=ERL                   :'Store the error and line number
370  IF ERTYPE%=17 THEN PRINT "ZOPEN ERROR!!":'If it is a "Z" series error, display it
380  PRINT "ERROR CODE=";ERTYPE%               :'Display the error code
390  PRINT "ERROR LINE=";ERLINE%               :'Display the error line
400  RESUME 320                                :'Return control to line 320
```

(executed in IBM PC/AT BASIC)

```
10   'Send data to the console (IBM PC/AT transmit side)
20   ON ERROR GOTO 380                        :'Branch to line 380 in case of error
30   '
40   '***** RS-232C CH.OPEN *****
50   DIM T%(8)                                :'Define an array
60   CH%=18                                   :'Define a communication channel
70   T%(0)=4800                               :'Set the baud rate
80   T%(1)=&H8                                :'Set the character length and parity bit
90   T%(2)=&H1                                :'Set the stop bit and DC control
100   T%(3)=0                                 :'Set signal control
110   T%(5)=0                                 :'Set the DC1 code and the DC code
120   T%(4)=0                                 :'Set the receive buffer
130   T%(6)=0                                 :'Set the DC3 code and the DC2 code
140   T%(7)=0                                 :'Set the DC4 code
150   ZOPEN #CH%,T%( )                        :'Open the communication channel
160   '
170   '***** RS CNTL *****
180   DIM CNTL%(1)                            :'Define an array
190   CNTL%(0)=32                             :'Set RS, ER, signal control
200   CNTL%(1)=&H1                            :'Set RS control ON
210   ZCNTL #CH%,0,CNTL%( )                   :'Execute signal control
220   '
230   '***** DATA SEND *****
240   DIM TBL%(2)                             :'Define an array
250   TBL%(0)=12                              :'Specify the required number of characters
260   TBL%(1)=0                               :'Store the number of bytes transmitted
270   TBL%(2)=100                             :'Set the timeout period
280   SD$="AD51H-BASIC"                       :'Define the data to transmit
290   ZSEND #CH%,0,TBL%( ),SD$                :'Transmit data
300   PRINT "SEND CHARACTER=";TBL%(1)         :'Display the number of characters
                                               transmitted
310   PRINT "SEND DATA=";SD$                  :'Display the data transmitted
320   '
330   '***** RS-232C CH.CLOSE *****
340   ZCLOSE #CH%                             :'Close the communication channel
350   END                                     :'Finish execution
360   '
370   '***** ERROR ROUTINE *****
380   ERTYPE%=ERR:ERLINE%=ERL                 :'Store the error and line number
390   IF ERTYPE%=94 THEN ERTYPE%=ZERROR(1)    :'If it is a "Z" series error, display it
400   PRINT "ERROR CODE=";ERTYPE%             :'Display the error code
410   PRINT "ERROR LINE=";ERLINE%             :'Display the error line
420   RESUME 340                              :'Return control to line 340
```

REMARK

See ZOPEN, ZSEND and ZCNTL. Also see section 7.3.4.

- Enables another program to use a resource that has been assigned a resource number.

Z RELEASE

SYNTAX	**ZRELEASE△[<resource number>]**	
	resource number	Specifies the resource number (in the range 0 31) assigned by the programmers to a resource used within a program.
EXAMPLE	**ZRELEASE 1**	The resource that has been assigned resource number one becomes available for use by other programs.

DESCRIPTION

- When the **ZRELEASE** command is executed by a program that previously executed a **ZRESERVE** command to obtain exclusive access to a resource, the specified resource is released, and becomes available for use by other programs.
- If when the resource is released, another program that also executed a **ZRESERVE** command is waiting to use the resource, the waiting program obtains exclusive access.
- The resource number assigned to the resource that is to be made available for use by other programs should be specified for **<resource number>.**
- If number is omitted, all resources currently reserved for exclusive access by the program are released.
- If the BASIC commands **SYSTEM, END** or **RUN** are executed, or if a program is changed while execution is temporarily suspended, all resources reserved for exclusive access by the program are released.

REMARKS

(1) For details on resource exclusive access management, see section 8.3 and the ZRESERVE command.

(2) See ZRESERVE.

- Makes it impossible for other programs to use a resource that has been assigned a resource number.

Z RESERVE

SYNTAX	**ZRESERVEΔ<resource number>[,<timeout period>]**

	resource number	Specifies the resource number (in the range 0 31) assigned by the programmers to a resource used within a program.
	timeout period	Specifies the maximum amount of time that the program will wait to reserve a resource. Specified as "HH:MM:SS:R" HH : hours (0 to 23) MM : minutes (0 to 59) SS : seconds (0 to 59) R : 100ms (0 to 9 indicates 0 to 900ms)
EXAMPLE	**ZRESEERVE 1,"0:0:5:0"**	The resource assigned resource number 1 is reserved, and cannot be used by other programs. If the resource cannot be reserved within five seconds, a timeout error occurs.

DESCRIPTION

- The **ZRESERVE** command makes it impossible for other programs executing simultaneously to access a resource that has been assigned a resource number by general consent of the programmers. When the **ZRESERVE** command is executed, the program may have exclusive access to the resource as detailed below.
- To cancel exclusive access of a resource, the **ZRELEASE** command should be executed using the same resource number as was specified with the **ZRESERVE** command.
- The **ZRESERVE** and **ZRELEASE** commands are used together so that a resource cannot be used by other programs for a desired time period or during a desired operation.
- The resource numbers to be assigned to each resource must first be decided on by the programmers. When writing programs, before the resource is used the **ZRESERVE** command should be executed, and after it is used, the **ZRELEASE** command should be executed.
- The resource number assigned to the resource which is to be reserved should be specified for **<resource number>**.

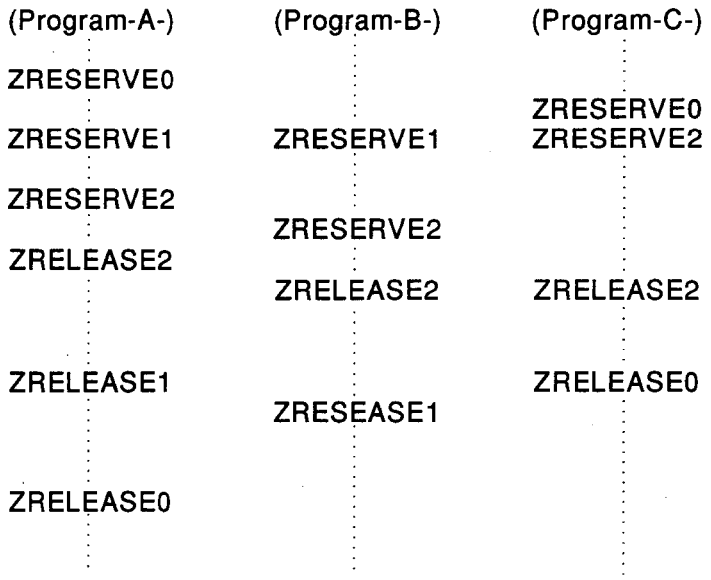
- **<Timeout period>** specifies the maximum amount of time the program will wait if the resource to be reserved is already in use by another program. (While waiting, program execution is suspended.) The time is set as follows:

  "HH:MM:SS:R"
  HH ... hours ("0" to "23")
  SS ... seconds ("0" to "59")
  MM ... minutes ("0" to "59")
  R ... 100ms ("0" to "9" indicates 0 to 900ms)

  If **<timeout period>** is omitted or if it is set to "0 ", the program will wait indefinitely for the resource.

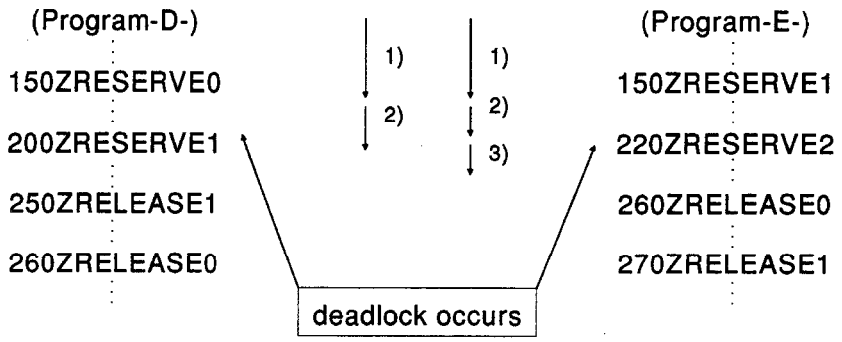
- It is possible for one program to reserve multiple resources at the same time.

- When several different resources are used simultaneously in several different programs, it is necessary to execute the **ZRESERVE** commands in a set order so that program deadlock does not occur (the programs become unable to execute), as shown below:
  (The following shows an example of **ZRESERVE** commands being executed in order from low resource numbers to high resource numbers.)



(Deadlock Example)

When programs D and E are executing simultaneously, when program D executes line 150, if program E already executed line 150, execution of program D is suspended at line 200.
When E executes line 220, since D has already reserved resource number zero, execution of program E is suspended at line 220.
Since D is waiting for resource number 1 and E is waiting for resource number 0, execution of both programs remains suspended.



The numbers in half brackets indicates the progress of program execution.

REMARKS

(1) For details on resource exclusive access management, see section 8.3.

(2) See ZRELEASE.

- Sends data from the communication port.

Z SEND

SYNTAX	**ZSENDΔ#<channel number>, 0, <control table>,<output element>**

	channel number	Specifies the communication port from which to send data.
	control table	Specifies parameters for sending data.
	output element	Specifies the variable or array in which the data to be sent is stored.
EXAMPLE	**ZSEND #1,0,A%( ),"TEST DATA"**	The character string "TEST DATA" is sent from CH1 (RS-232C) according to the parameters specified in integer array A%(0) and A%(2). The number of bytes of data sent is stored in A%(1).

DESCRIPTION

- The **ZSEND** command sends the data specified with **<output element>** from the communication port specified with **<channel number>** according to the parameters set with **<control table>**
- **<Channel number>** specifies which AD51H communication port to use. The channel numbers correspond to the communication ports as shown below:

Channel Number	Communication Port
1	CH1(RS-232C)
2	CH2(RS-232C)
3	CH3(RS-422)
4	CH4(PARALLEL)

Before executing the **ZSEND** command to send data from a communication port,the port must first be opened with the **ZOPEN** command. If the port is not opened,an error occurs.

- The **<control table>** is used to specify the number of bytes of data to send, and the timeout period. Also, the actual number of bytes of data sent is stored here.
An integer array is used for the **<control table>**.

[ ]%(0) ..... Specifies the desired number of bytes to send.

[ ]%(1) ..... The number of bytes sent is stored following completion of the transfer.

[ ]%(2) ..... Specifies the timeout period.

POINT

Although the <control table> uses only three array elements, the array must be defined with the DIM command. If the array is not defined with the DIM command,when the ZSEND command is executed, an error occurs. (Normally, if an array uses less than ten elements, it is not necessary to define it with the DIM command.)

(1) Number of bytes to send ..... [ ]%(0)

Specifies the number of bytes of data to send. Since the method of counting bytes depends on the contents of **<output element>**, refer to the following table:
If an integer variable is used as the output element, 2 bytes are sent.
If an integer array is used as the output element, 2 to 1024 bytes can be sent.
If an character array is used as the output element, 1 to 256 bytes can be sent.

(2) Timeout period ..... [ ]%(2)

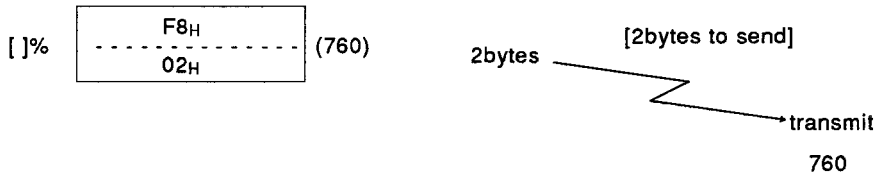
Specifies the wait time for sending data. When the **ZSEND** command is executed, if transmission of data is not completed within a specified time period, an error occurs. The timeout period is set to the specified value times 100ms.
If a timeout error occurs, the number of bytes of data sent before the timeout error is stored to [ ]%(1).
Values in the range 0 to 65535 may be specified, but when specifying values greater than 32767, hexadecimal representation should be used (decimal notation will cause an error when it is stored). If zero is specified, the timeout period is unlimited.

(3) Number of bytes sent ..... [ ]%(1)

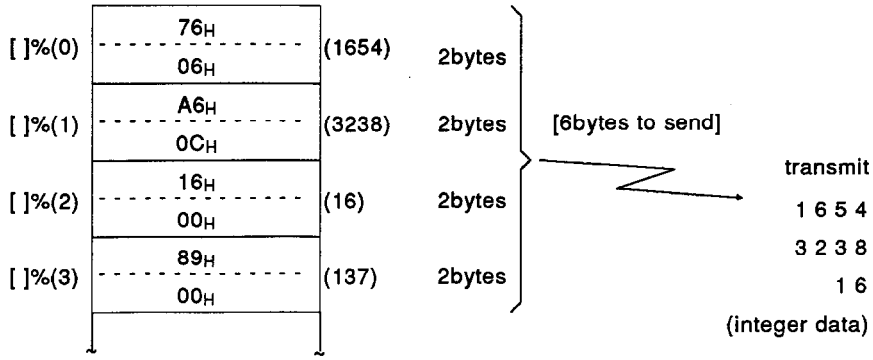
The number of bytes sent is automatically stored following completion of the data transmission. If data was sent normally, this value is the same as "number of bytes to send", so the value is used to check the condition of the data transfer.

- **<Output element>** is the storage area for the data that is to be sent. An integer variable, integer array, character string variable or character string array may be specified.

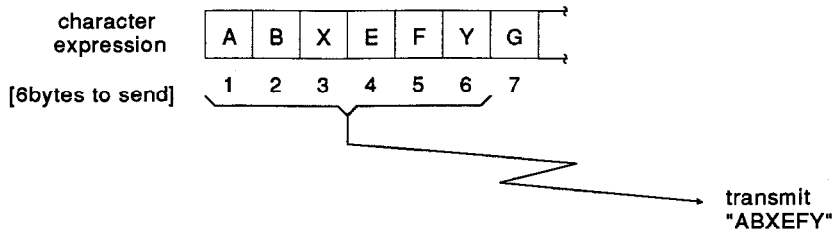
(1) When an **integer variable** is specified, one integer value can be sent. Since one integer value is handled as two bytes, the "number of bytes to send" should be set to 2.



(2) When an **integer array variable name** is specified, each element of the array contains two bytes, and the number of bytes of data specified with "number of bytes to send" can be sent. However, array element numbers cannot be specified,so be sure to store the data to send starting with element zero of the array.



(3) When a **character string variable** or character array is specified, each character in the string is one byte, and the number of bytes specified by "number of bytes to send" is sent.



- Before the **ZSEND** command is executed, the variable or array used as the **<output element>** should be defined with the **DIM** command if necessary and then the data to send should be stored in it.

> **POINT**
>
> **If an integer array or character string array is used for <output element>,even if the number of elements used by the array is less than 10, the array must be defined with the DIM command. If the array is not defined with the DIM command, when the ZSEND command is executed, an error occurs.**

- A channel number that has not been opened by the **ZOPEN** command cannot be specified. If a closed channel is specified, an error occurs.

- If the data specified for **<output element>** does not comprise as many bytes as "number of bytes to send", an error occurs, and the command is not executed.

- If the data specified for **<output element>** comprises more bytes than "number of bytes to send",only the number of bytes specified in "number of bytes to send"is sent. The remaining data is ignored.

PROGRAMMING EXAMPLE (executed in AD51H BASIC)

```
10  'Send data to the IBM PC/AT (AD51H transmit side)
20  ON ERROR GOTO 330                          :'Branch to line 330 in case of error
30  '
40  '***** RS-232C CH.OPEN *****
50  DIM T%(10)                                 :'Define an array
60  CH%=1                                      :'Define a communication channel
70  T%(0)=4800                                 :'Set the baud rate
80  T%(1)=&H8                                  :'Set the character length and parity bit
90  T%(2)=&H1                                  :'Set the stop bit
100  ZOPEN #A%,T%( )                           :'Open the communication channel
110  '
120  '***** PORT CNTL *****
130  DIM CNT%(10)                              :'Define an array
140  CNT%(0)=32                                :'RS, ER, signal control
150  CNT%(1)=&H1                               :'RS control
160  ZCNTL #CH%,0,CNT%( )                      :'Signal control
170  '
180  '***** DATA SEND *****
190  CIM TBL%(10)                              :'Define an array
200  TBL%(0)=30                                :'Specify the number of bytes to send
210  TBL%(1)=0                                 :'Store the number of bytes sent
220  TBL%(2)=100                               :'Set the timeout period
230  SD$="SOUSIN DATA:ABCDEFGH 12345678"       :'Define the data to send
240  ZSEND #CH%,0,TBL%( ),SD$                  :'Transmit
250  PRINT "SEND CHARACTER=";TBL%(1)           :'Display the number of characters sent
260  PRINT "SEND DATA=";SD$                    :'Display the data sent
270  '
280  '***** RS-232 CH.CLOSE *****
290  ZCLOSE #CH%                               :'Close the communication channel
300  END                                       :'Finish execution
310  '
320  '***** ERROR ROUTINE *****
330  ERTYPE%=ERR:ERLINE%=ERL                   :'Store the error and line number
340  IF ERTYPE%=17 THEN PRINT "ZOPEN ERROR!!":'If it is a "Z" series error, display it
350  PRINT "ERROR CODE=";ERTYPE%               :'Display the error code
360  PRINT "ERROR LINE=";ERTYPE%               :'Display the error line
370  REXUME 290                                :'Return control to line 290
```

(executed in IBM PC/AC BASIC)

```
10  'Receive data from the console (IBM PC\AC receive side)
20  ON ERROR GOTO 390                        :'Branch to line 390 in case of error
30  '
40  '***** RS-232 CH.OPEN *****
50  DIM T%(8)                                :'Define an array
60  CH%=18                                   :'Define a communication channel
70  T%(0)=4800                               :'Set the baud rate
80  T%(1)=&H8                                :'Set the character length and parity bit
90  T%(2)=&H1                                :'Set the stop bit and DC control
100  T%(3)=0                                 :'Set signal control
110  T%(4)=0                                 :'Set the receive buffer
120  T%(5)=0                                 :'Set the DC1 code and the DC code
130  T%(6)=0                                 :'Set the DC3 code and the DC2 code
140  T%(7)=0                                 :'Set the DC4 code
150  ZOPEN #CH%,T%( )                        :'Open the communication channel
160  '
170  '***** DATA RECEIVE *****
180  DIM TBL%(6)                             :'Define an array
190  TBL%(0)=30                              :'Specify the required number of characters
200  TBL%(1)=0                               :'Store the number of bytes received
210  TBL%(2)=300                             :'Set the timeout period
220  TBL%(3)=0                               :'Set the number of break characters
230  TBL%(4)=0                               :'Set break characters 1 and 2
240  TBL%(5)=0                               :'Set break characters 3 and 4
250  TBL%(6)=0                               :'Set break character 5
260  WOR%=TBL%(0)/2                          :'Read two bytes
270  DIM CT%(WOR%-1)                         :'Define an array
280  ZRECEIVE #CH%,0,TBL%( ),CT%( )          :'Receive data
290  PRINT "RECEIVE CHARACTER=";TBL%(1)      :'Display the number of characters received
300  FOR I%=0 TO WOR%-1
310  PRINT "RECEIVE DATA=&H";HEX$(CT%(I%))   :'Display the data received
320  NEXT I%
330  '
340  '***** RS-232C CH.CLOSE *****'
350  ZCLOSE #CH%                             :'Close the communication channel
360  END                                     :'Finish execution
370  '
380  ***** ERROR ROUTINE *****'
390  ERTYPE%=ERR:ERLINE%=ERL                 :'Store the error and line number
400  IF ERTYPE%=94 THEN ERTYPE%=ZERROR(1)    :'If it is a "Z" series error, display it
410  PRINT "ERROR CODE=";ERTYPE%             :'Display the error code
420  PRINT "ERROR LINE=";ERLINE%             :'Display the error line
430  RESUME 350                              :'Return control to line 350
```

REMARK

See ZOPEN, ZRECEIVE and ZCNTL. Also see section 7.3.4.

- Signals a specified event from a program.

Z SIGNAL

SYNTAX	**ZSIGNALΔ<event number>**	
	event number	Specifies the number (0 to 63) of the event to be signalled which was defined with the DEF ZEVENT command.
EXAMPLE	**ZSIGNAL 10**	The event defined as event number 10 is signalled.

DESCRIPTION

- The **ZSIGNAL** command signals a specified event from a program. If an event is first enabled by a program with the **ZEVENT** command, when the event is then signalled by the program with the **ZSIGNAL** command, execution of all programs waiting for the specified event is restarted.
- Execution of the program that signalled the event proceeds to the next statement regardless of the status of the programs waiting for the event.
- Events are defined with the **DEF ZEVENT** command and event release is enabled or disabled with the **ZEVENT** command. Also, to pause execution of a program until a specified event is released (to synchronize execution between programs), the **ZWAIT EVENT** command is used.
- In order for a program that is paused waiting for an event to be signalled to restart execution, an event with the same event number must be enabled within the program using the **ZEVENT** command. If the event is not enabled, even though the **ZSIGNAL** command is executed, control is not performed.

REMARKS

(1) For details on event processing, see section 8.2.

(2) See DEF ZEVENT, ZEVENT and ZWAIT EVENT.

- Starts a specified program.

Z START

SYNTAX	**ZSTARTΔ<number>** **ZSTARTΔ<number>,"[<drive number>:][<system name>\]<file name>"**

	number	Specifies the task number area number of the program to start.
	drive number	Specifies the memory card, floppy disk or hard disk where the program to be started is stored.
	system name	Specifies the system name under which the program to be started is stored.
	file name	Specifies the file name of the program to be started.
EXAMPLE	**ZSTART 5**	The program stored in task number area 5 is started.
	ZSTART 2,"0:PRO\TEST.BAS"	The program with file name "TEST.BAS" stored under system name "PRO" on the memory card in memory card slot 1 is read into BASIC task number area 2 and started.

DESCRIPTION

- The **ZSTART** command starts a program in a specified task number area.

STARTING A PROGRAM THAT IS STORED IN AD51H SYSTEM MEMORY

- If only **<number>**is specified, the program stored in the task number area indicated by **<number>**is started.
- In order for programs to be started this way, the specified task number area must be in one of the following conditions:
  (1) 'BOOT' has been set with the AD51H System Mode command "SET".
  (2) A program started by setting 'START' with the AD51H System Mode command "SET" is in the "not executing" condition due to execution of the **END** command.
  (3) A program started with the **ZSTART** command or with a sequencer interrupt is in the "not executing" condition due to execution of the **END** command.
- If there is no program stored in the task number area specified by an error occurs.

REMARK

For more details on AD51H multitasking operation and AD51H system mode, see the AD51H/AD51H-S3 User's Manual (Programming, Debugging volume).

READING A PROGRAM FROM A MEMORY CARD, FLOPPY DISK OR HARD DISK AND STARTING

- If "**<drive number>;<system name>\ <file name>**" is specified following **<number>**,a program is read from the memory card, floppy disk or hard disk into the task number area specified by and started.
- Only programs stored in the file area of a memory card can be read in. Programs stored in the execute area cannot be read in.
- The **<drive number>** specifications correspond to the physical devices as shown below:

  AD51H Memory Card 1 0

  AD51H Memory Card 2 1

  PC/AT FDD 2

  PC/AT HDD [C] 3
- **<System name>** specifies which system name the program is stored under. If the specified system name does not exist, a "File not found" error occurs.
- **<File name>** specifies the name and file extension of the file in which the program is stored.
- In order for programs to be started this way, the specified task number area must be in one of the following conditions:

  (1) 'BOOT' or 'ON' has been set with the AD51H System Mode command "SET".

  (2) A program started by setting 'START' with the AD51H System Mode command "SET" is in the "not executing" condition due to execution of the END command.

  (3) A program started with the **ZSTART** command or with a sequencer interrupt is in the "not executing" condition due to execution of the **END** command.
- If there is a program executing in the task number area specified by <event number> and 'OFF' has been set with the AD51H System Mode command "SET", an error occurs.

POINT

When a program is read into a task number area which contains a program in the "not executing" condition, the program is deleted.

- Changes the priority of a program.

Z URGENCY

SYNTAX	**ZURGENCYΔ<priority>**	
	priority	Specifies the execution priority level (0 to 10) of a program running under multitasking. As the priority number becomes smaller, the program's priority becomes higher.
EXAMPLE	**ZURGENCY 3**	The program's priority is set to 3.

DESCRIPTION

- The **ZURGENCY** command is used when BASIC programs are running under multitasking to change the execution priority level of the program that executes the command.
- Up to eight BASIC programs can be executed simultaneously on the AD51H. The AD51H operating system alternates execution of each program based on the programs' priority, thus effectively simulating parallel execution.
- When each program is started, its priority is 1.
- The **ZURGENCY** command's function is to change the priority of the program which executes it.
- When multiple programs are all in the "able to execute" condition, the "able to execute" condition, the **<priority>** number is used by the AD51H operating system as criteria for deciding which program to execute. If -1 is specified, it is considered the same as zero.
- The smaller the number specified for **<priority>**, the higher the program's priority becomes. (The higher a program's priority, the greater the chance it will be given permission to execute by the AD51H operating system.)
- Several programs may be set to the same priority, but if a number less than -1 or a number in the range 11 to 99 is specified for **<priority>**, an error occurs.
- If a number greater than 99 is specified for **<priority>**, the AD51H cannot guarantee the program's operation or results, so it is advisable not to set a priority number greater than 99.
  However, if a value greater than 99 is specified, an error does not occur.

REMARK

For details on task priority, see sections 8.1 and 8.4.

- Suspends program execution until a specified time period has elapsed.

Z WAIT DELAY

SYNTAX	**ZWAITΔDELAYΔ<time period>**

	time period	Specifies the time period to suspend execution in the format "HH:MM:SS:R" HH : hours (0 to 23) MM : minutes (0 to 59) SS : seconds (0 to 59) R : 100ms (0 to 9 indicates 0 to 900ms)
EXAMPLE	**ZWAIN DELAY "00:01:00:0"**	Program execution is suspended for one minute.

DESCRIPTION

- The **ZWAIT DELAY** command suspends program execution until the time period specified with **<time period>** has expired. (The program enters the "waiting" condition.)
- **<Time period>** specifies the length of time that program execution will be suspended, and is set as shown below:

  "HH:MM:SS:R"

  HH : hours ("0"to"23") MM : minutes("0"to"59")

  SS : seconds("0"to"59") R : 100ms ("0"to"9" indicates 0 to 900ms)
- The program's execution is not restarted until after the specified time period has expired.

- Suspends program execution until a specified event is signalled.

Z WAIT EVENT

SYNTAX	**ZWAITΔEVENTΔ<event number> [,<timeout period>]**

	event number	Specifies the event number assigned to the event to be awaited when it was defined.
	timeout period	Specifies the maximum length of time the program can wait for the specified event to be signalled. Specified in the format "HH:MM:SS:R" HH : hours (0 to 23) MM : minutes (0 to 59) SS : seconds (0 to 59) R : 100ms (0 to 9 indicates 0 to 0ms)
EXAMPLE	**ZWAIT EVENT 1**	The program's execution is suspended until event number 1 is signalled.
	ZWAIT EVENT 3,"00:01:00:0"	The program's execution is suspended until event number 3 is signalled. If the program waits longer than one minute and event three still has not been signalled, a timeout error occurs.

DESCRIPTION

- The **ZWAIT EVENT** command causes program execution to be suspended until the event specified by **<event number>** is signalled. (The program enters the "waiting" condition.)
- **<Event number>** specifies the number assigned to the event which is to be awaited when the event was defined.
- **<Timeout period>** specifies the maximum length of time the program will wait for the specified event to be signaled. It is set as shown below: (While the program is waiting, execution is suspended.)

  "HH:MM:SS:R"
  HH : hours ("0"to"23") MM : minutes("0"to"59")
  SS : seconds("0"to"59") R : 100ms ("0"to"9" indicates 0 to 900ms)

  If **<time period>** is omitted, or if "0:0:0:0" is specified, the program willwait indefinitely.
- The **ZWAIT EVENT** command is used to synchronize execution between the program that executes it and the program that signals the event.

- In order to restart execution of the program that that executed the **ZWAIT EVENT** command, one of the other programs must enable release of the event by executing the **ZEVENT** command.
- The event to be signalled must be defined with the **DEF ZEVENT** command.
- Several programs can wait for the same event at the same time.

REMARK

See ZEVENT, ZSIGNAL and DEF ZEVENT.

12. CREATING BASIC PROGRAMS USING A COMPILER

This section explains how to create a BASIC program using a compiler.

> When using interpreter BASIC, it is not necessary to read this section.
> Use compiler BASIC after reading this section and confirming restricted items, etc.

12.1 Differences Between Compiler BASIC and Interpreter BASIC

An AD51H-S3 can use both compiler BASIC and interpreter BASIC.

The difference in execution between compiler BASIC and interpreter BASIC is as follows:

Compiler BASIC ..... Translates programming language into machine language before the program is run, and then an AD51H-S3 executes the machine language program.

Interpreter BASIC ..... Translates programming language into machine language and runs it at the same time while an AD51H-S3 is executing the program.

Neither is objectively better or worse: both have advantages and disadvantages as shown below.

	Advantages	Disadvantages
Compiler BASIC	• Fast execution .	• Debugging is difficult. • Many detailed restrictions.
Interpreter BASIC	• Debugging is easy.	• Slower execution.

Select compiler or interpreter according to the purpose to be fulfilled.

12.2 Flow When Creating a Program Using a Compiler

This section how to create a BASIC program, how to translate it using a compiler, and how to execute it using an AD51H-S3.

1) Start the software package.

.....Start the SW1IX-AD51HPE type software package.

2) Create a BASIC program by online programming.

3) Execute it by using an interpreter, and confirm the operation.

.....Confirm whether it operates in the restricted range of a compiler.

4) Store the program to the hard disk by using the SAVE instruction.

.....Store the program to the C drive of the hard disk (designated drive number 3).

5) Terminate the software package.

.....The screen returns to the DOS command line.

6) Compiler execution

.....Input and compile DBC <file name>.

7) Confirm the executable program (~.EXE) file.

C:\AD51H\USR>DIR/W

.....Confirm whether the ~.EXE file has been created after executing the compilation.

8) Restart the software package.

.....Start the SW1IX-AD51HPE type software package.

9) Select online programming from the menu and transfer to the system mode.

10) Use the MSAVE command to save it to the execution area of the memory card.

.....MSAVE <task ID>,, file name

11) Use the SET command to change the start condition to START.

.....Change the start condition from BOOT to START.

12) Execution

.....To execute in the programming mode, input GO R[,D].

To execute in the execution mode, set the mode setting switch 1 to 0 or 1, and reset the AD51H-S3.

12.3 Necessary Items For Compiling

An assembler and a linker are necessary for compiling programs created by an AD51H-S3. But, since they are not included in the SW1IX-AD51HPE software package, they must be purchased separately.

The recommended assembler and linker are shown below.

- Mitsubishi recommends the following:

  Microsoft Macro Assembler Ver.4.0 or 5.1 (supporting AX)

Since linkers are not sold separately, use a linker (Microsoft linker) after Version 3.5 such as the linker contained in MASM or MS-DOS.

12.4 Registering Assemblers and Linkers to a Hard Disk

This section explains how to register assemblers and linkers to a hard disk.

Only those files (MASM.EXE, LINK.EXE) that are necessary for a compilation by using an AD51H-S3 are copied.

1) Start

↓

2) Insert the Microsoft MASM assembler disk in drive A.

↓

3) Use the MS-DOS COPY command to copy MASM.EXE to the hard disk.

   C:\>COPY A:\MASM.EXE C:\[Enter]

↓

4) After copying is completed, take out the floppy disk, and insert the MASM utility disk in drive A.

↓

5) Use the MS-DOS COPY command to copy LINK.EXE to the hard disk.

   C:\>COPY A:\LINK.EXE C:\[Enter]

↓

6) After copying is completed, take out the floppy disk.

↓

7) Completed

12.5 Starting the Compiler

It is necessary to start the compiler to compile a created program. Compilation can be executed by inputting DBC<file name> name on MS-DOS.

How to set the compiler format and the available options are shown below.

DBC [-4] [-v] [-w-] [-d] Source file name [.BAS] [Loading file name.EXE]

Source file name	BASIC source file name. If an extension has not been designated, the source file name extension is considered to be '.BAS'.
Loading file name.EXE	Compiled loading file name. If a file name has not been designated, the source file name extension is replaced by '.EXE', and the file name becomes a loading file name.
-4	Designated when using Ver. 4.0. If this has not been designated, Ver 5.1 is considered to be used.
-v	States during compiling are fully displayed.
-w-	Warnings are not displayed.
-d	A program is compiled to check the following errors when executing an operation. (For debugging) • When adding, subtracting, or multiplying an integer, errors are checked for. If an overflow is detected, an Overflow error is generated. • When executing an operation, the ranges of the values of the array's subscripts are checked. If a value is out of range, a Subscript out of range error is generated. However, if this option is designated, the size of the execution program is enlarged, and the execution speed decreases.

The following example shows the flow for compiling (file name: COMP.EXE) a program (file name: INTER.BAS) created by an interpreter.

```
1) C:\>DBC -v INTER.BAS COMP.EXE [Enter]
                                .......Start compilation. Designate -v option.
   C:\>c:

   C:\>cd\ad51h\usr

2) C:\AD51H\USR>c:\ad51h\system\dbc -Lc:\ad51h\system -v
   INTER.BAS COMP.EXE



   BASIC COMPILER Ver 1.0
                                .......Start the compiler.
   masm $1.asm;
   Microsoft (R) macro Assembler Version 5.10
   Copyright (C) Microsoft Corp 1981, 1988. All right reserved.


       xxxxx Bytes symbol free


           0 Warning Errors
           0 Severe Errors

3) Link c:\ad51h\system\dbb.obj $1,comp,/map,c:\ad51h\
   system\dbc.1ib
                                .......Start a linker.
   Microsoft (R) Segmented-Executable Linker Version 5.01.20
   Copyright (C) Microsoft Corp 1984-1988. All rights reserved.
   Definitions File[NUL.DEF]:[Enter]
                                .......Input the ENTER key.
                                       This is not always displayed.
   LINK: warning L4021: no stack segment
                                .......Ignore this error.
   C:\AD51H\USR>

4) C:\AD51H\USR>                .......Compilation completed.
```

12.6 Precautions When Compiling

(1) Compiling

Always use a Microsoft Macro Assembler when compiling, since programs cannot be compiled by using any other assembler.

(2) Workpiece filed created by the compiler

Since, the following file names are files created by the BASIC compiler, the user cannot utilize them.

$n.ASM	assembler source file "n" indicates the numerical.
$n.OBJ	object file
BC.TMP	workpiece file
BASIC$$$.INC	include file

(3) Errors during compiling

Errors during compiling are stored in the assembler source file $n.ASM and also displayed on the screen. Confirm errors by referring to this file.

(4) Program operation confirmation

Before compiling a program, execute it using an interpreter, and confirm proper operation. Since programs cannot be edited after compiling, if an error occurs after compiling, that program must be modified by an interpreter and must be compiled again.

(5) Errors when the DBC compiler is used

If the DBC compiler is used, LINK: warning L4021:no stack segment is displayed during the link. However, this message does not create any difficulty in the operation of a compiled program. Therefore, ignore this error.

(6) Variable sizes

The interpreter consumes only a character-string area of the length of a character and the compiler always consumes 256 bytes per variable. Therefore, use the CLEAR instruction to secure a large number of variable storage areas when using a great many variables.

(7) Order in which an expression is executed

To increase execution speed, the compiler optimizes expressions. Therefore, the priority level of the expression and the combination rule do not change. However, the order in which items in an expression are executed may not be the same.
For example, in ASC(INKEY$)-ASC(INKEY$)*2, it cannot be judged which is executed first. If the result of an expression differs from that of the interpreter, execute the expression after dividing it and storing the middle value to the variable.

(8) Integer operation upgrading

If the middle result of an expression is not in the range of an integer in the addition, subtraction, multiplication, or division of an integer, it is upgraded automatically to a real number by the interpreter, and the calculation is executed. However, the operation is executed in the range of the integer by the compiler. In this case, use the CSNG or CDBL function to convert an integer value into a real number.

12.7 Execution Using an AD51H-S3

To execute a compiled program using an AD51H-S3, the compiled program must be registered in the execution area of a memory card.
The procedure for registration to the execution area of a memory card is shown below.

MSAVE <Task ID>[,[V],"< File name>" [,[Location]] or

MSAVE <Task ID>[,V]

Task ID	BASIC task ID that is stored in the execution area 1 ≤ task ID ≤ 8
V	Specify verification. After writing is completed, the contents of the main memory and the memory card are verified.
"File name"	Specify the compiled file name (~.EXE) that is read to the main memory.
Location	Specify the position where a task is allocated. 0, 16, 32,...368 (Increased by 16) If it is abbreviated, the allocation is automatically executed.

Notes

- The 'Error: Location' error, which shows that a location cannot be allocated, will sometimes occur when MSAVE is executed. When this happens, either designate a vacant location and execute MSAVE or set the start conditions of all tasks to OFF and execute MSAVE.
- The 'System: code =824' error, which shows that there is no work area, will sometimes occur when MSAVE is executed. When this happens, set the start conditions of all tasks to OFF, and execute MSAVE of all tasks again.

For instance, when saving a compiled file (example: COMP.EXE) by using MSAVE to position 32 of task ID 1, the procedure is as follows:

```
1) S>MSAVE 1,,"3:COMP.EXE",32 [Enter]
   |                          .......Saving a file to the execution area of task 1.
   | SAVE(Y/N)? Y             .......Select Y.
   | SAVE OK
   | S>                       .......Completed.
2) S>SET 1,START [Enter]
   |                          .......Changing the start condition from 'BOOT'
   |                                 to 'START'.
   | SET OK
   | S>
3) S>GO R [Enter]             .......Transferring to the execution mode.
```

12.8 Instructions and Functions

12.8.1 Compilability of instructions and functions

The compilability of instructions and functions is shown in Table 12.1 below.

o: Compilable

Δ: Compilable with restrictions

x: Non-compilable

Table 12.1 Compilability of instructions and functions

Instructions and Functions	Compilability	Remarks	Page/Section
ABS	o		P11-2
ASC	o		P11-3
ATN	Δ	With restrictions	S12.8.2-1,
AUTO	x	Non-compilable	S12.8.2-2,
BEEP	o		P11-6
BIN$	Δ	With restrictions	S12.8.2-3,
BSWAP	o		P11-9
CDBI	o		P11-11
CDBL	o		P11-13
CHAIN	x	Non-compilable	S12.8.2-4
CHR$	o		P11-17
CIDB	o		P11-18
CINT	o		P11-20
CISN	o		P11-22
CLEAR	x	Non-compilable	S12.8.2-5
CLOSE	o		P11-25
CLS	o		P11-26
COM ON/OFF/STOP	o		P11-27
COMMON	x	Non-compilable	S12.8.2-6
CONSOLE	o		P11-29
CONT	x	Non-compilable	S12.8.2-7
COS	Δ	With restrictions	S12.8.2-8
CSNG	o		P11-32
CSNI	Δ	With restrictions	S12.8.2-9
CVD	o		P11-35
CVDMBF	o		P11-36
CVI	o		P11-37
CVS	o		P11-38

(continued)

Table 12.1 Compilability of instructions and functions (continued)

Instructions and Functions	Compilability	Remarks	Page/Section
CVSMBF	o		P11-39
DATA	Δ	With restrictions	S12.8.2-10
DATE$	o		P11-43
DEFDBL	Δ	With restrictions	S12.8.2-11
DEF FN	Δ	With restrictions	S12.8.2-12
DEFINT	Δ	With restrictions	S12.8.2-13
DEFSNG	Δ	With restrictions	S12.8.2-14
DEFSTR	Δ	With restrictions	S12.8.2-15
DEF ZEVENT	o		P11-54
DELETE	x	Non-compilable	S12.8.2-16
DIM	Δ	With restrictions	S12.8.2-17
END	o		P11-60
EOF	o		P11-61
ERASE	x	Non-compilable	S12.8.2-18
ERL	o		P11-63
ERR	o		P11-64
ERROR	o		P11-65
EXP	Δ	With restrictions	S12.8.2-19
FIELD	o		P11-67
FILES	x	Non-compilable	S12.8.2-20
FIX	o		P11-69
FOR to NEXT	Δ	With restrictions	S12.8.2-21
FORMAT	o		P11-72
FRE	Δ	With restrictions	S12.8.2-22
GET	o		P11-75
GETMEM	o		P11-76
GOSUB RETURN	Δ	With restrictions	S12.8.2-23
GOTO	o		P11-83
HEX$	Δ	With restrictions	S12.8.2-24
IF GOTO ELSE	o		P11-86
IF THEN ELSE	o		P11-88
INKEY$	o		P11-90
INPUT	Δ	With restrictions	S12.8.2-25
INPUT$	o		P11-94
INPUT#	o		P11-97
INSTR	o		P11-98
INT	o		P11-100

(continued)

Table 12.1 Compilability of instructions and functions (continued)

Instructions and Functions	Compilability	Remarks	Page/Section
KEY	o		P11-102
KEYLIST	x	Non-compilable	S12.8.2-26
KILL	o		P11-104
LEFT$	o		P11-106
LEN	o		P11-108
LET	o		P11-109
LFILES	x	Non-compilable	S12.8.2-27
LINE INPUT	Δ	With restrictions	S12.8.2-28
LINE INPUT#	o		P11-114
LIST	x	Non-compilable	S12.8.2-29
LLIST	x	Non-compilable	S12.8.2-30
LOAD	x	Non-compilable	S12.8.2-31
LOC	o		P11-119
LOCATE	o		P11-120
LOF	o		P11-122
LOG	Δ	With restrictions	S12.8.2-32
LPRINT	o		P11-124
LPRINT USING	o		P11-125
LSET	o		P11-126
MERGE	x	Non-compilable	S12.8.2-33
MID$(No.1)	o		P11-130
MID$(No.2)	o		P11-132
MKD$	o		P11-133
MKDMBF$	Δ	With restrictions	S12.8.2-34
MKI$	o		P11-136
MKS$	o		P11-138
MKSMBF$	Δ	With restrictions	S12.8.2-35
NAME	o		P11-142
NEW	x	Non-compilable	S12.8.2-36
OCT$	Δ	With restrictions	S12.8.2-37
ON COM GOSUB	Δ	With restrictions	S12.8.2-38
ON ERROR GOTO	o		P11-149
ON GOSUB	o		P11-151
ON GOTO	o		P11-153
OPEN	o		P11-155
PCRD	o		P11-157
PCWT	o		P11-188

(continued)

Table 12.1 Compilability of instructions and functions (continued)

Instructions and Functions	Compilability	Remarks	Page/Section
PRINT	o		P11-299
PRINT USING	Δ	With restrictions	S12.8.2-39
PRINT#	o		P11-235
PRINT# USING	Δ	With restrictions	S12.8.2-40
PUT	o		P11-237
PUTMEM	o		P11-238
RDSET	Δ	With restrictions	S12.8.2-41
READ	Δ	With restrictions	S12.8.2-42
REM	o		P11-246
RENUM	x	Non-compilable	S12.8.2-43
RESTORE	x	With restrictions	S12.8.2-44
RESUME	Δ	With restrictions	S12.8.2-45
RIGHT$	o		P11-252
RND	o		P11-253
ROT	Δ	With restrictions	S12.8.2-46
RSET	o		P11-256
RUN(No.1)	x	Non-compilable	S12.8.2-47
RUN(No.2)	o		P11-259
SAVE	x	Non-compilable	S12.8.2-48
SEARCH	o		P11-262
SGN	o		P11-264
SHA	Δ	With restrictions	S12.8.2-49
SHT	Δ	With restrictions	S12.8.2-50
SIN	Δ	With restrictions	S12.8.2-51
SPACE$	o		P11-270
SPC	Δ	With restrictions	S12.8.2-52
SQR	Δ	With restrictions	S12.8.2-53
STOP	Δ	With restrictions	S12.8.2-54
STR$	o		P11-274
STRING$	o		P11-275
SYSTEM	x	Non-compilable	S12.8.2-55
SWAP	o		P11-278
TAB	Δ	With restrictions	S12.8.2-56
TAN	Δ	With restrictions	S12.8.2-57
TIME$	o		P11-281
TROFF	x	Non-compilable	S12.8.2-58
TRON	x	Non-compilable	S12.8.2-59

(continued)

Table 12.1 Compilability of instructions and functions (continued)

Instructions and Functions	Compilability	Remarks	Page/Section
VAL	Δ	With restrictions	S12.8.2-60
WHILE WEND	Δ	With restrictions	S12.8.2-61
WIDTH	o		P11-289
WTSET	Δ	With restrictions	S12.8.2-62
ZBAS	o		P11-293
ZCLOSE	o		P11-294
ZCNTL	o		P11-295
ZEVENT	o		P11-313
ZIDV	o		P11-314
ZLDV	o		P11-315
ZMESSAGE	o		P11-316
ZMESSAGE CLOSE	o		P11-319
ZMESSAGE GET	o		P11-320
ZMESSAGE KILL	o		P11-323
ZMESSAGE OPEN	o		P11-324
ZMESSAGE PUT	o		P11-325
ZMOVE	o		P11-327
ZODV	o		P11-332
ZOPEN	o		P11-334
ZRECEIVE	o		P11-336
ZRELEASE	o		P11-341
ZRESERVE	o		P11-342
ZSEND	o		P11-345
ZSIGNAL	o		P11-350
ZSTART	Δ	With restrictions	S12.8.2-63
ZURGENCY	o		P11-353
ZWAIT DELAY	o		P11-354
ZWAIT EVENT	o		P11-355

12.8.2 Different instruction and function specifications when using a compiler

When executing an operation using a compiler, the specifications are different from the specifications when an interpreter is used.

This section explains these differences. Instructions other than instructions explained in this section have specifications that are the same as for an interpreter. Therefore, for those instructions, see the interpreter explanations.

Table 12.2 Different instruction and function specifications when using a compiler

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
1	ATN	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-4
2	AUTO	• The AUTO instruction is unusable. • AUTO can be used as a variable name.	———————	11-5
3	BIN$	• If a value outside the range (-32768 to 65535) is designated to the expression, the operation result is the same as when 32767 is designated.	• Before using the BIN$ function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-7
4	CHAIN	• The CHAIN instruction is unusable. • When a compiler is used, a "not support" error occurs.	• Substitute a RUN (No.2) instruction. • Make sure the RUN (No.2) instruction does not have the following CHAIN instruction function. • Execution at the place of interruption by designating a line number: Use GETMEM and PUTMEM instructions to deliver numerical values between programs, and use the ON GOTO instruction to execute a jump to the desired line number according to its numerical value. • Variable transfer using the ALL option: Use the GETMEM and PUTMEM instructions to execute the transfer.	11-15
5	CLEAR	• The CLEAR instruction is ignored. • When a compiler is used, it is ignored.	• When clearing a variable, use an assignment instruction.	11-24
6	COMMON	• The COMMON instruction is unusable. • When a compiler is used, a "not support" error occurs.	• Use the GETMEM and PUTMEM instructions to execute the transfer.	11-28

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
7	CONT	• The CONT instruction is unusable. • When a compiler is used, a "not support" error occurs.	———————	11-30
8	COS	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-31
9	CSNI	• The CSNI instruction does not check overflow.	• Before using the CSNI function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-33
10	DATA	• Double quotation marks (") in the DATA instruction can be used as the symbols to enclose character-string constants. • If there is no double quotation mark (") symbol at both end of a character-string constant, the range from the head to the (") symbol, or the range from the (") symbol to the end of the character-string constant is considered data.	• Use the double quotation marks (") correctly.	11-41
11	DEFDBL	• Definition is executed before execution statement. • A variable already declared in DEFINT, DEFSNG, DEFDBL, and DEFSTR instruction cannot be declared again to a different type by using another instruction.	• Execute declarations before execution statements. • Do not execute re-definitions.	11-45
12	DEFFN	• When defining or calling a function, do not put a blank between FN and the name. • The type designation must always be added to the variable in the name, dummy argument, and function definition of the DEFFN instruction. • When calling other user-defined functions in the function definition expression, they must be defined before they are called. • Once defined, a user function cannot be redefined.	• Add the type designation. • When using other user-defined functions, define them before calling them.	11-47

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
13	DEFINT	• Definition is executed before execution statement. • A variable already declared in DEFINT, DEFSNG, DEFDBL, and DEFSTR instruction cannot be declared again to a different type by using another instruction.	• Execute declarations before execution statements. • Do not execute re-definitions.	11-49
14	DEFSNG	• Definition is executed before execution statement. • A variable already declared in DEFINT, DEFSNG, DEFDBL, and DEFSTR instruction cannot be declared again to a different type by using another instruction.	• Execute declarations before execution statements. • Do not execute re-definitions.	11-51
15	DEFSTR	• Definition is executed before execution statement. • A variable already declared in DEFINT, DEFSNG, DEFDBL, and DEFSTR instruction cannot be declared again to a different type by using another instruction.	• Execute declarations before execution statements. • Do not execute re-definitions.	11-52
16	DELETE	• The DELETE instruction is unusable. • DELETE can be used as a variable name.	———	11-57
17	DIM	• A variable cannot be used for the numeric expression that designates the size of an array by using the DIM instruction. • During execution, array subscript ranges are not checked. (However, execution of a check can be designated by using option [-d] for debugging during compiling.)	• Designate the maximum size.	11-58
18	ERASE	• The ERASE instruction is ignored. • When executing a compilation, a warning occurs.	• When using this to define a new array, define the maximum size array and use the array again. • When using this to erase an array, delete the ERASE instruction.	11-62
19	EXP	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-66
20	FILES	• The FILES instruction is ignored. • When executing a compilation, a warning occurs.	———	11-68

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
21	FOR ~ NEXT	• FOR and NEXT instructions must always have a one-to-one correspondence.	• Give the FOR and NEXT instructions one-to-one correspondence.	11-70
22	FRE	• The FRE function always returns to 0. • When executing a compilation, a warning occurs.	———	11-74
23	GOSUB ~ RETURN	• "RETURN without GOSUB" is not checked.	• Use a counter to count and check the GOSUB and RETURN instructions.	11-81
24	HEX$	• If a value outside the range (-32768 to 65535) is designated to the <numeric expression>, the operation result is the same as when 32767 is designated.	• Before using the HEX$ function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-84
25	INPUT	• When executing an input operation using the INPUT instruction, this does not have a screen edit function. • When a numerical value is input, overflow is not checked. Therefore, when too large a value is input, an error will not occur, and it becomes a negative number. • When the number of items divided by ',' is not the same as the number of variables, 'Redo from start' is displayed, and the INPUT instruction is executed again.	• Separate it into another task and execute the input operation using an interpreter.	11-92
26	KEYLIST	• The KEYLIST instruction is ignored. • When executing a compilation, a warning occurs.	———	11-103
27	LFILES	• The LFILES instruction is ignored. • When executing a compilation, a warning occurs.	———	11-111
28	LINE INPUT	• When executing an input operation using the INPUT instruction, this does not have a screen edit function.	• Process only part of the LINE INPUT instruction by using another interpreter task.	11-112
29	LIST	• The LIST instruction is unusable. • LIST can be used as a variable name.	———	11-115
30	LLIST	• The LLIST instruction is unusable. • LLIST can be used as a variable name.	———	11-117
31	LOAD	• The LOAD instruction is unusable. • LOAD can be used as a variable name.	———	11-118

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
32	LOG	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-123
33	MERGE	• The MERGE instruction is unusable. • When a compiler is used, a "not support" error occurs.	———	11-128
34	MKDMBF$	• Only double precision internal representation IEEE format data is converted. When data of other formats is input, it is considered as IEEE format data and is converted.	———	11-135
35	MKSMBF$	• Only single precision internal representation IEEE format data is converted. When data of other formats is input, it is considered as IEEE format data and is converted.	———	11-140
36	NEW	• The NEW instruction is unusable. • NEW can be used as a variable name.	———	11-144
37	OCT$	• If a value outside the range (-32768 to 65535) is designated to the <numeric expression>, the operation result is the same as when 32767 is designated.	• Before using the OCT$ function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-145
38	ON COM GOSUB	• An interrupt is executed at the head of each instruction by the interpreter, but an interrupt is executed at the head of the line by the compiler.	• Do not describe a multiple statement on the first line of interruption processing.	11-147
39	PRINT USING	• One PRINT USING instruction can describe eight units of display data.	• Split display data into several PRINT USING instructions.	11-231
40	PRINT# USING	• One PRINT# USING instruction can describe eight units of display data.	• Split display data into several PRINT# USING instructions.	11-236
41	RDSET	• Array subscripts and bit ranges are not checked.	• Before using the RDSET function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-243

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
42	READ	• Even if the value exceeds the real number maximum value when reading a decimal constant, an overflow does not occur, and the real number maximum value is returned. • Even if the value exceeds the integer maximum value when reading octal and hexadecimal constants, an overflow does not occur, and the integer maximum value is returned. • When a values defined in the variable types of the READ and DATA instructions do not agree, a 'Syntax error' can occur on the READ instruction side. • However, this error does not always occur. Example: 10 DATA &H000012 20 READ A! 'An error does not occur. 30 DATA &H12X 40 READ A! 'An error occurs. • When an error occurs, data next to the data that caused the error is read.	• Try not to execute an abnormal read.	11-245
43	RENUM	• The RENUM instruction is unusable. • RENUM can be used as a variable name.	———	11-247
44	RESTORE	• Line number 0 is unusable.	———	11-249
45	RESUME	• Execution is restarted in a line unit. • When a multiple statement is used, pay attention to the following: RESUME: Execution is restarted from the first number of the line where it occurred. RESUME NEXT: Execution is restarted from the head of the following line. RESUME line number: Execution is restarted from the head of the designated line.	• Split multiple statements into several lines.	11-250
46	ROT	• The ROT function does not check an overflow.	• Before using the ROT function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-254
47	RUN (No.1)	• The RUN (No.1) instruction is unusable.	———	11-258
48	SAVE	• The SAVE instruction is unusable. • SAVE can be used as a variable name.	———	11-260

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
49	SHA	• Argument overflows are not checked.	• Before using the SHA function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-265
50	SHT	• Argument overflows are not checked.	• Before using the SHT function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-267
51	SIN	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-269
52	SPC	• When the SPC function is put at the end of the PRNT instruction, a line feed is executed.	• Add ';' (semicolon) after the SPC function.	11-271
53	SQR	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-272
54	STOP	• A STOP instruction terminates a program. (Operates the same as an END instruction) • When executing a compilation, a warning occurs.	———	11-273
55	SYSTEM	• The SYSTEM instruction is unusable. • SYSTEM can be used as a variable name.	———	11-277
56	TAB	• When the TAB function is put at the end of the PRNT instruction, a line feed is executed.	• Add ';' (semicolon) after the TAB function.	11-279
57	TAN	• When a double precision real number is included in the <numeric expression>, a double precision value is returned. In other cases, a single precision value is returned.	• Enclose the <numeric expression> by using the CSNG function, and always make the numerical value single precision.	11-280
58	TROFF	• The TROFF instruction cannot be used. Even when a compilation is executed, errors are not displayed. Therefore, reliable operations when a compilation is executed cannot be guaranteed.	———	11-283
59	TRON	• The TRON instruction cannot be used. Even when a compilation is executed, errors are not displayed. Therefore, reliable operations when a compilation is executed cannot be guaranteed.	———	11-284

(continued)

Table 12.2 Different instruction and function specifications when using a compiler (continued)

No.	Instructions and Functions	Different Specifications, Restrictions, and Precautions	Alternative Plans	Pages
60	VAL	• Double precision values are always returned.	• After using the VAL function for conversion, substitute and use it for the variable of a necessary type.	11-285
61	WHILE ~ WEND	• WHILE and WEND instructions must always have a one-to-one correspondence.	• Give the FOR and NEXT instructions one-to-one correspondence.	11-287
62	WTSET	• Array subscripts and bit ranges are not checked.	• Before using the WTSET function, use the IF instruction to check the range, and generate an error using the ERROR instruction.	11-290
63	ZSTART	• If the multitask setting of the task designated to the <number> is IP, the interpreter program is started. If the multitask setting is CP, a compiler program is started. • The file designated to the <file name> must be a file (~.EXE) that is created by DBC (the BASIC compiler), if the task designated to the <number> of the ZSTART instruction is compiler BASIC. If a file other than a file created by DBC is designated, an error occurs or the AD51H-S3 system malfunctions. • A task that has been executed one time can be restarted by designating a file name using the ZSTART instruction.	• When restarting a task, designate the loading file name (~.EXE) of the compiler program.	11-351

APPENDICES

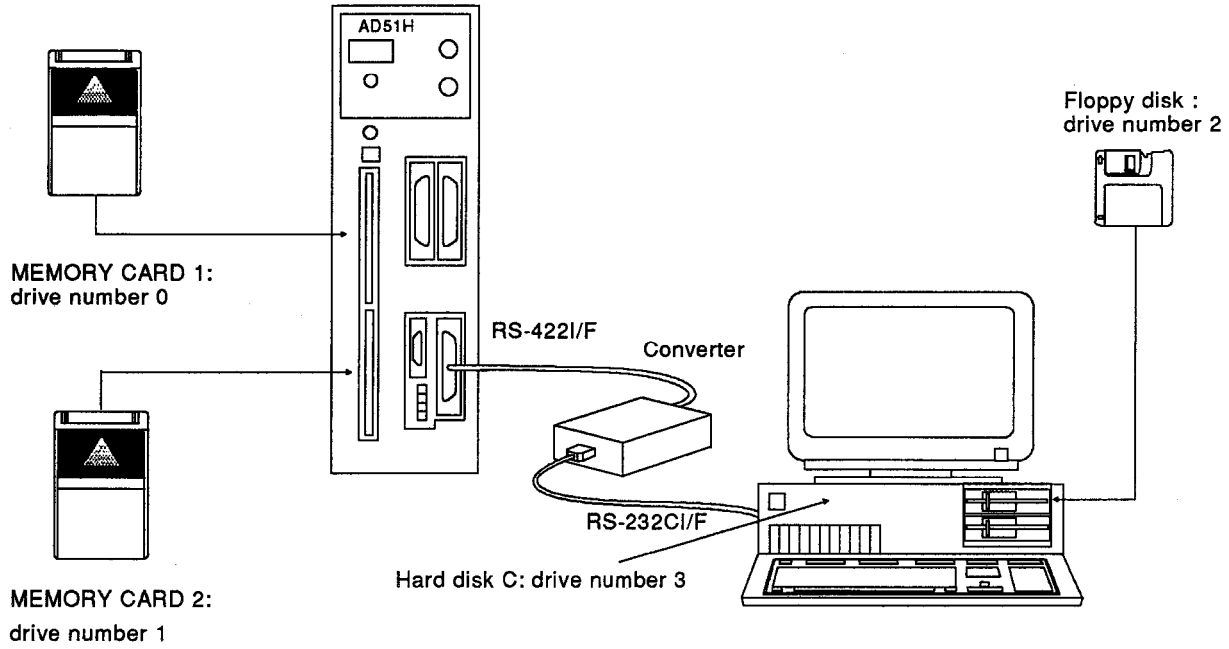
APPENDIX 1 FILE NAMES

When storing a BASIC program or data in a memory card or a FD, specify the file name as follows:

"[Drive number]:[System name]\[File name]"

1.1 Drive Numbers

Specify the drive number for the memory card interface, FD, and HD used to store and read a program and data. The memory card interfaces, an FDD, and HDD are allocated with a number. Specify this number for the drive number.



- When using the SW[ ]IX-AD51HPE software package and offline programming function, the drive number changes as shown below.

  FDD of the PC/AT ..... A

  HDD of the PC/AT C

  The memory card interface ..... Cannot be used.

  The SW[ ]IX-AD51HPE Operating Manual gives details.

- When only specifying a drive number by using the FILES command, add a colon (:) after the drive number.

1.2 System Names

Use a system name to divide programs and data in a memory card, FD, or HD into groups.The system name must conform to the following rules:

Characters that can be used:

Alphabet	(A-Z and a-z)
Numbers	(0-9)
Special symbols	(!, #, $, %, &, ', (,), -, @, ^, _, {, }, ~)

The number of characters that can be used:

Up to 8 characters

System names that cannot be used:

Since the following words all have specific meanings, they cannot be used for system names:

AUX, CLOCK, CON, NUL, PRN, BAT, COM, EXE

However, these words can be used as a part of a system name.

Example

AUX ... NG

AUXNO ..... OK
TOAUX ..... OK
TOAUXNO ... OK

REMARKS

(1) When specifying a drive number and system name by using the FILES command, make sure to add an "\" symbol after the system name.

(2) The same system name cannot be used more than once in the same memory card, FD, or HD.

(3) The system name used in AD51H-BASIC is similar to the subdirectory used in MS-DOS. However, a system name cannot be made under a system name.

1.3 File Names

The file name is a name given to a program and data in a memory card, an FD, or an HD.

The file name consists of the file name and an extension.

These must follow the following rules.

Difference of the name of a file and an extension:

Divide the name of a file and an extension with a period.

Name of a file · Extension

Characters that can be used:

Alphabet	(A-Z and a-z)
Numbers	(0-9)
Special symbols	((!, #, $, %, &, ', (,), -, @,^, _, {, },~)

The number of characters that can be used:

Name of a file	Up to 8 characters
Extension	Up to 3 characters

System names that cannot be used:

Since the following words all have specific meanings,they cannot be used as file names:

AUX, CLOCK, CON, NUL, PRN, BAT, COM, EXE

However, these words can be used as part of a file name.

Example

AUX ... NG

AUXNO ..... OK
TOAUX ..... OK
TOAUXNO ... OK

POINT

The same file name cannot be used more than once in the same system name.

1.4 Wildcards

Use a wildcard to display or delete several file names in a memory card, FD, or HD.A wildcard is only valid for a file name.
A wildcard can be used with all characters.Two kinds of wildcards - with an asterisk (*) and with a question mark (?) - are used.

- Asterisk (*)
  This symbol is unrelated to the type of character or number of characters.

Example	Description
A*.*	All file names that begin with A
*.BAS	All file names with a BAS extension
.	All file names in the specified system name

POINT

The character specified after an asterisk (*) in the name of a file and an extension is ignored.

***Z* is considered as *.*.**
***.*1 is considered as *.*.**

- Question mark (?)
  This symbol can be used for all types of characters.

Example	Description
???123.BAS	All file names that have 1, 2, and 3 in the fourth digit, fifth digit, and sixth digit and have a BAS extension
TEXT?.*	All file names that have TEXT at the beginning of the file name and one character after that

REMARK

Because the * symbol and the ? symbol function like the joker in a deck of cards, they are called wildcards.

1.5 Precautions When Using Wildcards

Wildcards have the following limitations:

1) The wildcard can be used with the following commands:

   FILES, LFILES, KILL

2) A question mark (?) expresses a one-byte character.

3) If a question mark (?) is specified for the last part of a file name and the extension, any file name with the same character preceding the question mark and the same extension also becomes valid.

   Example

   AD51H??.BAS AD51HAB.BAS
   AD51HG.BAS
   AD51H.BAS

   All these file names become valid.

4) A character after an asterisk (*) is ignored.

POINT

Before using a wildcard with the KILL command, confirm the file that become valid by using the FILES command.
A file erased by a KILL command cannot be restored.

1.6 How to Specify a File Name

For efficient file management, use the following method to make a file name. Wildcards are very useful for making file names.

1) Attach the same extension to data to be used with the same program.

2) Attach a BAS extension to all program files.

3) All related program file names and data must begin with the same character.
   (Even if the same characters are used in the latter half of the file names, an asterisk cannot be used as a wildcard.)

APPENDIX 2 PRECAUTIONS WHEN DOING INTERRUPTION PROCESSING

By using AD51H-BASIC, interruption procssing can be started as follows:

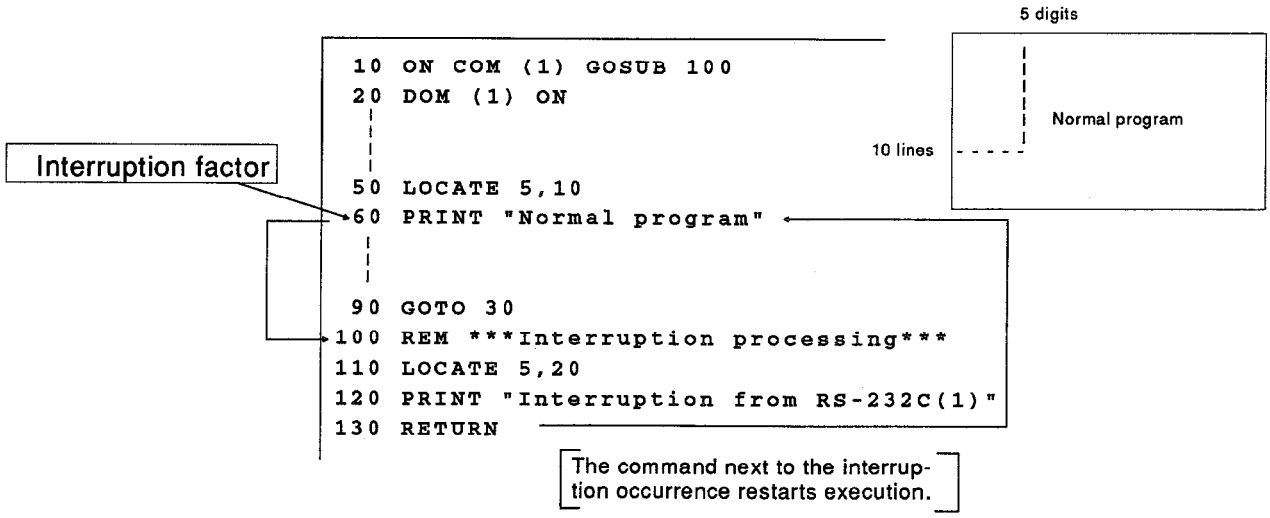
1) Start a subroutine by an interrupt from the communications apparatus connected to the AD51H interface. (ON COM GOSUB command)

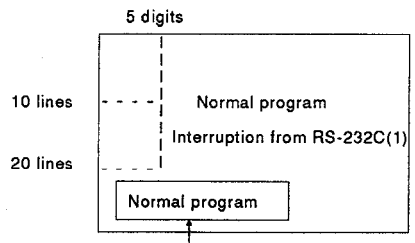
2) If an error occurs, execute an error processing routine. (ON ERROR GOTO command)

If either of the above factors occurs, the interruption procesinloop executes interruption processing.
Thus, the execution position of the interruption processing loop in the program cannot be predetermined.
Therefore, if interruption processing takes place when a character display is being executed or during console switching using a ZIDV or ZODV command, the character display position and console switching setting sometimes become invalid (See the following example.)
Create a program so that character displays and console switcing are not done during interruption processing.

Example

When a character display is executed during interruption processing





Because the PRINT command at the 120th line inserted after the LOCATE command at the 110th line, the PRINT command at the 60th line is executed here.

APPENDIX 3 COMMANDS AND FUNCTIONS TO SWITCH PROGRAMS EXECUTED BY MULTITASKING

If the following commands and functions are used, a program to be executed by multitasking is switched.

Command	Command	Command	Command	Built-in functions
AUTO		PRINT#		
	KILL	PRINT#USIG		
CHAIN				
		PUT	ZEVENT	
CLS	LFLES	PUTMEM		FRE
	LINE			
CONT	LINE INPUT	RUN		EOF
	LINE INPUT#		ZEVENT DEF	
	LIST	SAVE	ZRECEIVE	
	LLIST		RELEASE	INKEY$
END	LOAD	SET	ZRESERVE	INPUT$
	LOCATE		ZSEND	
FILES	LPRINT	STOP	ZSIGNAL	
	LPRINT USING		ZSTART	
GET		SYSTEM	ZURGENCY	
GETMEN			ZWAIT DELAY	
	MERGE	TIME$	ZWAIT EVENT	
		TRON	ZSEND	
INPUT	NAME	TROFF	ZRECIEVE	PCRD
INPUT#	NEW			PCWT
KEY				
KEYLIST				
	PRINT			
	PRINT USING			

POINT

Using the above commands and functions independently of specification by the ZURGENCY command switches the program being executed.

APPENDIX 4 CODE TABLES

4.1 Character Code Table

Code of the higher 4 bits (hexadecimal)	Code of the lower 4 bits (hexadecimal)																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E	SP	0	@	P		p	═	═	SP	—			α	
1	S_H	D_1	!	1	A	Q	a	q	═	═	°				β	
2	S_X	D_2	"	2	B	R	b	r	═	═	⌈				ε	
3	E_X	D_3	#	3	C	S	c	s	═	═	⌋				μ	
4	E_T	D_4	$	4	D	T	d	t	═	═	、				φ	
5	E_Q	N_K	%	5	E	U	e	u	═	═	.				θ	
6	A_K	S_N	&	6	F	V	f	v	═	═					∞	
7	B_L	E_B	'	7	G	W	g	w	═	═					Ω	
8	B_S	C_N	(	8	H	X	h	x	═	═					Σ	
9	H_T	E_M	)	9	I	Y	i	y	═	═					π	
A	L_F	S_B	*	:	J	Z	j	z	═	═					↑	
B	H_M	E_C	+	;	K	[	k	{	═	═					↓	
C	C_L	→	,	<	L	\	l	|	═	═					▶	
D	C_R	←	-	=	M	]	m	}	═	═					+	
E	S_O	↑	.	>	N	^	n	~	═	═						
F	S_I	↓	/	?	O	—	o	DEL	═	═						

POINT

The character in ▒ varies with the terminal used (the appropriate console manual gives details).

REMARK

How to read a code table
To get a character code for a character, find a corresponding number in the horizontal column (higher 4-bit code in hexadecimal), and then, find a corresponding number in the vertical column (lower 4-bit code in hexadecimal).

Example

A is represented as 41 (hexadecimal).
This value is represented in hexadecimal. To represent this value in decimal, it becomes as follows:

$$4 \times 16^{1} + 1 \times 16^{0} = 65$$

Therefore, the character code for A is 65 (&H41).

4.2 List of Control Keys

Key Operation	Function
[Enter](Return)	Ends data entry of a line and returns the cursor to the first column of the next line or area.
[→],[←],[↑],[↓]	The cursor moves in the direction of the arrow key.
[Insert]	The character at the cursor position and all following characters move one character to the right.
[Delete]	The character to the left of the cursor is deleted, and all following characters move one character to the left.
[Break]	Program execution is interrupted, and the BASIC command is received.
[Esc]	The Esc code (CHR $ (& H1B)) is input.
[F1] ~ [F10]	A character string defined for each function key is input.
[Tab]	A horizontal tab is inserted.
[Print Screen]	The contents of the screen are output to the printer.
[Back Space]	Works the same as the [Delete] key.
[Home]	The cursor moves to the home position (upper left hand corner of the screen).
[Ctrl]+[B]	The cursor moves one character to the left.
[Ctrl]+[C]	Works the same as the [Break] key.
[Ctrl]+[E]	All characters from the cursor position to the end of the line are cleared.
[Ctrl]+[G]	A buzzer (beep) sounds.
[Ctrl]+[H]	The character to the left of the cursor is erased. Function is the same as the [Delete],and [Back Space]keys.
[Ctrl]+[I]	The cursor moves to the next tab position. The tab position is set for every eight characters. Function is the same as the [Tab] key.
[Ctrl]+[J]	A line feed (LF code) is inserted.
[Ctrl]+[K]	The cursor moves to the home position (upper left hand corner of the screen). Works the same as the [Home] key.
[Ctrl]+[L]	The text screen is cleared.
[Ctrl]+[M]	Works the same as the Return key ([Enter]).
[Ctrl]+[N]	The cursor moves one character to the right.
[Ctrl]+[R]	The character at the cursor position and all following characters move one character to the right.Works the same as the [Insert] key.
[Ctrl]+[S]	The screen display is temporarily interrupted.

4.3 Screen Display Control Codes When Using a General-Purpose console

List of display control codes

Function	Description	Code (ASCII)		BASIC command
		When setting VG-620	When setting VT-382/VT-220	
Line feed	Return and line feed	CR and LF code (0DH, 0AH)	CR and LF code (0DH, 0AH)	———
Screen clear	Erasure of all screens	ESC + E (45H)	ESC + [(5BH) + 2(32H) + J(9AH)]	CLS
XON	A transfer enable from an external device is specified.	DC1 code (11H)	DC1 code (11H)	———
XOFF	A transfer disable from an external device is specified.	DC3 code (13H)	DC3 code (13H)	———
Escape	Escape, sequence, and introducer	ESC code (1BH)	ESC code (1BH)	———
Cursor control	Cursor and backward	BS code (08H)	BS code (08H)	———
	Cursor: Upward	ESC + A (41H)	ESC + [(5BH) + 1(31H) + A(41H)]	———
	Cursor: Downward	ESC + B (42H)	ESC + [(5BH) + 1(31H) + B(42H)]	———
	Cursor: Rightward	ESC + C (43H)	ESC + [(5BH) + 1(31H) + C(43H)]	———
	Cursor: Leftward	ESC + D (44H)	ESC + [(5BH) + 1(31H) + D(49H)]	———
Cursor addressing	The cursor position is controlled by absolute addressing.	ESC + Y (59H) + line designation code (20H-9FH) + column designation code (20H-9FH)	ESC + [(5BH) + line position designation, (3BH) + column position designation + H(48H)	LOCATE
Audible alarm	A bell sounds.	BEL code (07H)	BEL code (07H)	———

4.4 Lists of Error Messages and Error Codes

If an error occurs while an AD51H-BASIC command is being executed, the following happens:

During online programming

..... An error message appears on the console screen.

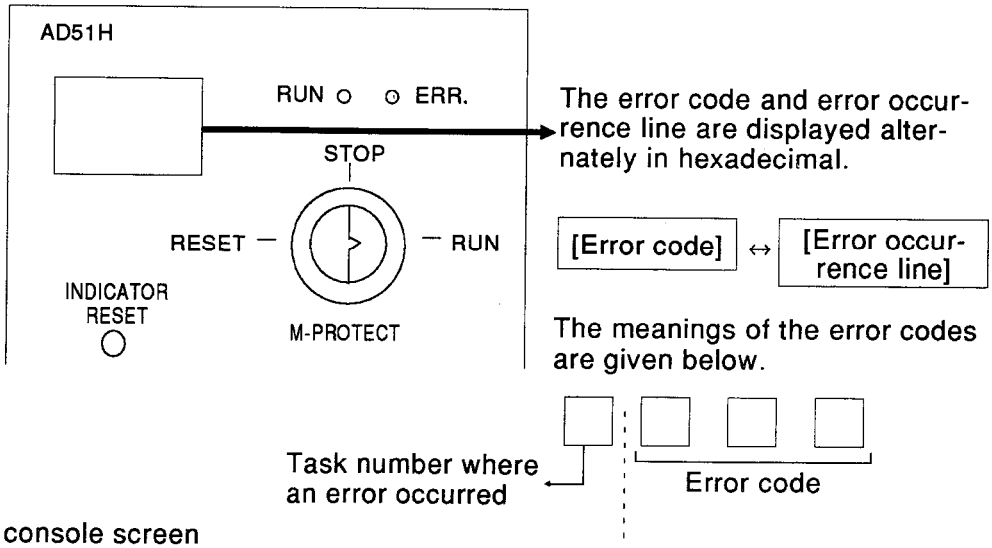
During online programming in the edit mode

```
[Error message]    in    [Error occurrence line]
OK ..................................... Waiting for a command to be input.
■
```

During online programming in the edit mode

..... The LED indicator on the front of the AD51H and the console screen indicate an error.

LED indicator



console screen

```
[Error message]    in    [Line number]
S>■ ..................................... The AD51H system mode becomes valid.
```

POINT

If an error processing routine is defined by the ON ERROR GOTO command during online programming when an error occurs, an error processing routine is executed.
No error message is displayed.

REMARK

The ON ERROR GOTO command part of Section 9 gives details about the error processing routine.

4.4.1 Error messages

This section lists the error messages as well as their codes, meanings, and corrective actions.
See Appendix 4.4.2 when any of the following error messages is displayed.

System errors (nnnn) in xxxx are found in Appendix 4.4.2.

The codes in the table are codes given to the EER function or error codes displayed on the LED.

Error message	Code (Hexadecimal)	Meaning	Corrective Actions
Bad drive number	65(41H)	The drive designation was incorrect. A drive number that does not exist in the system was designated.	Designate a correct drive number.
Bad File Data	25(19H)	Data format on the file was incorrect.	Check the data file attribute . Check if it is a program file.
Bad file mode	54(36H)	The file mode was incorrect.	Check the sequential file access file mode.
Bad file name	62(3EH)	The file name designation was incorrect.	Designate a correct file name.
Bad file number	52(34H)	A file number was designated outside the range of 1 to 8.	Designate a file number within the correct range.
Can't continue	17(11H)	Execution could not be restarted using the CONT command. (The pointer value is destroyed.)	Execute it from the beginning.
Device already open	75(4BH)	The ZOPEN command was executed for a port already opened by the ZOPEN command.	Modify the program so the ZOPEN command is not executed twice.
Device I/O error	74(4AH)	The ZOPEN command designation was incorrect. (For details, refer to ZOPEN command.)	Designate the ZOPEN command correctly.
Device not open	76(4CH)	No device was opened.	Execute the ZOPEN command.
Direct Statement in file	63(3FH)	When an ASCII format file was read, it contained a direct statement.	Check the file because a program without a line number was read.
Disk full	60(3CH)	• Writing was attempted to a full disk. • 128 or more files were written.	Delete any unnecessary files.
Disk I/O Error	57(39H)	• The file area of a designated memory card was set to write disabled. • There was no file area in the designated memory card. • An error occurred during the disk I/O.	• Cancel the memory card write disabled. • Format a file area in the memory card. • This is a fatal error and program processing cannot be continued.

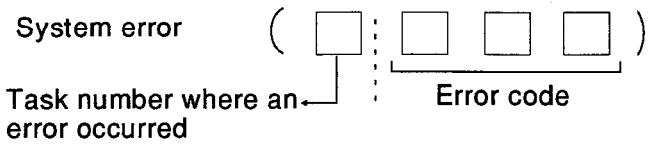
Error message	Code (Hexadecimal)	Meaning	Corrective Actions
Division by zero	11(B_H)	Division by 0 was executed, and the value of the result overflowed.	Modify the program so 0 division cannot be done.
Feature not implimented	26(1A_H)	There was no program.	Execute a program after doing MSAVE.
FIELD overflow	50(32$_H$)	An area of 256 bytes or more was designated by the FIELD command.	Designate the area within 255 bytes or less.
File already exists	58(3A_H)	A file name that was attempted to be changed by the NAME command already exists.	Either change the existing file name or the file name whose change was attempted.
File already open	55(37$_H$)	OPEN and KILL, etc. were executed for an already opened file.	Modify the program not to execute OPEN, etc.. Only execute them after the file is closed.
File not found	53(35$_H$)	A file name was designated that did not exist in the memory card, floppy disk, hard disk, etc.	Designate a correct file name.
File not OPEN	71(47$_H$)	An attempt was made to refer to a file that was not opened.	Modify the program to refer to the file after opening.
Illegal direct	12(C_H)	An attempt was made to execute a command that cannot be used in the direct mode.	Use a command that can be utilized in the direct mode.
Illegal function call	5(5$_H$)	A function was called incorrectly.	Use the correct form to designate the function.
Input past end	61(3D_H)	An input command was executed after reading all data in the file.	Modify the program not to do an input command.
Line buffer overflow	23(17$_H$)	A character was input outside the range of characters that can be input on one line.	Input within 255 characters or less.
Missing operand	22(16$_H$)	A parameter needed in the statement was not designated.	Designate the necessary parameter.
Next without FOR	1(1$_H$)	The FOR command and the NEXT command did not correspond correctly. (There are a great many NEXT commands.)	Modify the program so the FOR command and the NEXT command correspond correctly.
No RESUME	19(13$_H$)	Execution of a program could not be continued because there was no RESUME command in the error processing routine.	Execute a RESUME command.
Out of data	4(4$_H$)	Though there was no data to be read, the READ command was executed.	Use the DATA command to designate data.
Out of memory	7(7$_H$)	The memory capacity was too small. (Either the program was too big or all stacks were used.)	Either shorten a variable or GOSUB command subroutine.
Out of string space	14(E_H)	There was no memory capacity in which a character string could be stored.	Use the CLEAR command to enlarge the memory area.

Error message	Code (Hexadecimal)	Meaning	Corrective Actions
Overflow	6(6H)	The operation result or input numerical value exceeded the allowable range.	Set it within the allowable range.
Position not on Screen	24(18H)	The designated cursor position was outside the screen range.	Set the LOCATE command within the range.
Redimensioned arrey	10(AH)	Two definitions were designated to an array or user function.	Do not designate two definitions.
Rename across disks	68(44H)	An attempt was made to execute renaming between different drives using the NAME command.	Execute renaming within the same drive.
RESUME without error	20(14H)	There is a RESUME command although control was not transferred to the error processing routine.	Either remove the RESUME command or create an error processing routine.
RETURN without GOSUB	3(3H)	The GOSUB command and the RETURN command did not correspond correctly. (There are a great many RETURN commands.)	Modify the program so that the GOSUB command and the RETURN command correspond correctly.
Sequential after PUT	69(45H)	An attempt was made to access a sequential file after executing the PUT command.	Modify the program not to access random files as sequential files.
Sequential I/O Only	70(46H)	Did not execute except for sequential I/O.	Execute sequential file access only.
String formula too complex	16(10H)	A character expression was too complex.	Make the character expression less complex.
String too long	15(FH)	The number of characters of one character variable exceeded 255.	Make the number of characters 255 or less.
Subscript out of range	9(9H)	The value of an arrangement variable subscript was outside the specifed range.	Set the subscription value not to exceed the range defined by the DIM command.
Syntax error	2(2H)	The command description had a syntax error.	Correct any syntax errors.
System error (nnnn)	-(-)	See Appendix 4.4.2.	———
Time out error	31(1FH)	The designated number of byte characters was not input in the time designated by the INPUT$ command.	Make sure it is input in the designated time.
Type mismatch	13(DH)	Variable types didn't match in the left or right parts of an expression or in the argument of a function.	Match the variable types correctly.
Underfined line number	8(8H)	A line number (jump destination such as GOTO and GOSUB) neccessary for a jump destination did not exist.	Make sure there is a jump destination line number.
Underfined user function	18(12H)	An attempt was made to quote a user function that was not defined by the DEFFN function.	Use a user function after defining it by utilizing the DEFFN function.

Error message	Code (Hexadecimal)	Meaning	Corrective Actions
WEND without WHILE	78(4E$_H$)	The WHILE command and WEND command did not corresponding correctly. (There are great many WENDs.)	Make sure the WHILE command and the WEND command correspond correctly.
WHILE without WEND	77(4D$_H$)	The WHILE command and WEND command did not corresponding correctly. (There are great many WHILEs.)	Make sure the WHILE command and the WEND command correspond correctly.
Unprintable error	Undefined (-)	Undefined message error.	———

4.4.2 Table of system error codes

A system error code is represented by four hexadecimal digits.
The meanings are explained below:



POINT

The error code given by the ERR function is in decimal.
Convert this into hexadecimal number by using the HEX$ function.
Then, consult the error code table.

800H and above ..... General errors

Error Codes	Meanings	Corrective Actions
803H	The ON COM GOSUB command was registered twice.	Register the ON COM GOSUB command only once.
804H	The designated date and time were incorrect.	Set the correct time, date, and week.
805H	The CPU did not have a clock function.	Use a CPU with a clock function.
807H	The ZSTART command could not be executed for a designated task.	Execute the task which was stopped.
908H	There was not enough memory.	Make sure the execution area of the memory card is large enough.
809H	The memory of a designated task cannot be ensured. The size was not designated.	Either make the sizes of each task small or reformat the memory card again to ensure a large enough execution area. Use the SET command to designate the size.
910H	A debug command was executed for a compiler BASIC task.	Do not execute a debug command for a compiler BASIC task.
80AH	Execution could not be done in the edit mode.	Do not change execution from the debug mode for a task whose start condition is OFF.
80BH	The number of kanji IN/kanji OUT code characters is incorrect.	Correct the number of characters.
80CH	A CPU side error was detected when the date and time were designated.	Check the CPU.
828H	An attempt was made to utilize an extension function reserved for future use.	This function cannot be used.
830H	A message port number was incorrect.	Set a correct message port number.
831H	A designated message port was already defined.	Since the message port was already defined, it is not necessary to redefine it.
832H	A designated message port was not yet defined.	Use the message port after defining it.
833H	A message length exceeded 256 bytes.	Set the message length within 255 bytes or less.
834H	A message length designation was incorrect.	Set the message length correctly.
835H	A designated message port was not closed.	Close the message port.
836H	A designated message port was not opened.	Open the message port.
841H	A designated address was outside the range. The first device number when accessing ED was incorrect.	Set the address within the range. Set the first device number correctly.
842H	A transfer number of bytes designation was incorrect. The number of device points processed when accessing ED was not set correctly.	Set the transfer number of bytes correctly. Set the number of device points processed correctly.
881H	A task which was forcibly ended during processing was aborted because of an error.	Check if there is a place for forcible interruption in the program. Ensure that [Ctrl] + [C] cannot be input.

Error Codes	Meanings	Corrective Actions
8AEH	A designated interface could not be used.	Change the interface.
8C3H	There was no area in which a message could be written.	Either enlarge the message area or delete any unnecessary messages.
8CDH	An attempt was made to write 16 or more data items to one message port.	Reduce the size of the message to be written.

900H and above ..... File processing errors

Error Codes	Meanings	Corrective Actions
902H	A designated file was not found.	Designate the file name correctly.
903H	A designated system name was not found.	Designate the system name correctly.
905H	Improper processing was done for a memory card, floppy disk, etc.	Cancel the write protect.
90BH	An attempt was made to save a file using the MSAVE command which was not an execute form file created by the BASIC compiler.	Designate a file compiled by using a Microsoft Macro Assembler.
983H	FILES and LFILES commands were executed from more than one task.	Modify the program so as not to execute FILES and LFILES commands from more than one task.
984H	A fatal error occurred in the memory card.	Replace the memory card.
985H	The file area of a memory card was not formatted correctly.	Correctly format the file area of the memory card.
9C1H	A memory card was not installed. A memory card was faulty. A memory card install request switch was set to OFF.	Install a memory card. Replace the memory card. Switch the memory card install request switch to ON.
9C2H	A memory card was write-protected.	Cancel the write protect.
9C3H	A memory card was not formatted correctly.	Reformat the whole memory card.
9E4H	The memory card header information was incorrect.	Reformat the whole memory card.
9E5H	The size designated in the memory card format was different from the actual size of the memory card.	Match the size to be designated in the memory card format with the actual size of the memory card.
9E6H	Writing was executed to a ROM or ROM memory card.	Set write disabled.
9E8H 9E9H	A designated area did not exist in the memory card.	Check the memory card.

A00H and above . . . . . Communications processing errors

Error Codes	Meanings	Corrective Actions
A2AH A2BH A61H A62H A73H	Abnormal communications with a PC/AT.	Reset the AD51H-S3.
A75H	Memory card verification error	Confirm the size, etc. of the memory card.
A82H	A designated interface could not be used.	Change the interface.
A83H	Time out error	Confirm the transmission side device.
A84H	Receive error	Read the cause of the receive error on processing code 64 of the ZCNTL command, and confirm the error description.
A85H	A communication buffer overflowed.	Do not send so much data.
A91H	A request character number was not designated correctly.	Designate the request character number correctly.
A92H	A communication buffer size was not designated correctly.	Set the communication buffer size within the range of 1024 bytes or less.
A93H	A baud rate was not designated correctly.	Designate the baud rate correctly.
A94H	A character-string length was not designated correctly.	Designate the character-string length correctly.
A95H	A parity bit was not designated correctly.	Designate the parity bit correctly.
A96H	A stop bit was not designated correctly.	Designate the stop bit correctly.
A97H	The number of break characters was not designated correctly.	Designate the number of break characters correctly.
AA1H	The parallel interface printer was abnormal.	Check the printer.
AC1H	The size of the arrangement used as a control table was not sufficient.	Enlarge the size of the arrangement.
AC2H	An attempt was made to receive more data than the input element. An attempt was made to transmit more data than the output element.	Receive data that is the same as an input element. Transmit data that is the same as an output element.
AC4H	During communications, a request number of bytes was too large.	Set the correct request number of bytes.
AC7H	The ZCNTL command processing code is incorrect.	Check the processing code.

B00H and above ..... PC CPU processing errors

Error Codes	Meanings	Corrective Actions
B10H	The station of a designated PC number did not exist.	Check the designated station.
B11H	Communication between an AD51H-S3 and a PC CPU was faulty.	Check the AD51H-S3 and PC CPU.
B12H	A special-function module was not installed at a designated I/O address.	Make sure there is a special-function module at the designated I/O address.
B13H	A step number (address) was designated which exceeded the parameter range.	Set the step number (address) within the parameter range.
B18H	A remote RUN/STOP was done from other modules.	Cancel the remote RUN/STOP from other modules.
B20H	The MELSECNET (II) was not operating correctly.	Check the MELSECNET (II).
B21H	A designated special-function module could not be accessed.	Check if the special-function module is faulty.
B40H	Time out error	Check the CPU.
B41H	An error occurred during CPU data communications.	Confirm if the CPU caused an error.
B46H	A designated CPU is abnormal.	Confirm if the CPU caused an error.
B80H	The contents of the control tables of the PCRD and PCWT commands are incorrect. The data storage area was too small.	Confirm the control table contents. Enlarge the data storage area.
B81H	An attempt was made to execute processing that cannot be done by the designated CPU.	Designated processing cannot be performed by using a corresponding CPU.
B82H	A device name (device code) was incorrect.	Designate a correct device code.
B83H	A device number was incorrect.	Designate an available device number.
B84H	Though monitoring registration was not executed, an attempt was made to execute monitoring.	Execute monitoring after monitoring registration.
B85H	During monitoring registration, a different PC number was set.	Set the PC number that is the same as during monitoring registration.
B85H	During monitoring registration, a different system configuration was used.	Use the system configuration that is the same as during monitoring registration.
B87H	The number of points to be monitored is excessive.	Decrease the number of points to be monitored.
B88H	Block number processing of the designated extension file register cannot be executed.	Change it to an extension file register that can be executed.
B89H	An attempt was made to change a program or a parameter during ROM operations.	Do not change programs and parameters during ROM operations.
B8AH	Processing that cannot be done during CPU RUN was attempted.	Execute processing after making a CPU STOP.
B8DH	The processing code of a PCRD or PCWT command was incorrect.	Designate a correct processing code.

F00H and above ..... Warning error

Errors	Meanings	Corrective Actions
F80H	Low voltage in MEMORY CARD 1 battery	Replace the battery when the memory card is connected and the power supply of the AD51H-S3 is turned ON.
F81H	Low voltage in MEMORY CARD 2 battery	Replace the battery when the memory card is connected and the power supply of the AD51H-S3 is turned ON.

4.4.3 Error messages when compiling

This section explains the error messages that can occur during compiling. Error messages are classified into the following three types:

(1) Fatal error

This error immediately stops compilation.
A fatal error includes errors in connection with files, memory shortage errors, and errors inside the compiler.

[Sample error display]

```
FATAL --- file I/O error
```

When a fatal error is detected, the compiler immediately stops. Remove the cause of the error and reexecute the compilation. When a compiler stops compiling, the "compiler aborted" message is displayed.

(2) Error

This error indicates syntax errors in BASIC programs, violations of restrictions, and locations that cannot be correctly compiled.

[Sample error display]

```
;;10 A$=12345
;;        ^
---syntax error in string expression
```

After detecting as many errors as possible, the compiler stops compiling. Reexecute compilation after removing the error causes and confirming the operation by using an interpreter. When a compiler stops compiling, the "compiler aborted" message is displayed.

(3) Warning

This error message indicates a problem position, which is not actually an error. A typical warning is the detection of a statement to be ignored by the compiler.

[Sample error display]

```
;;10 CLEAR
;;    ^
```

When a warning is displayed, the compiler does not stop compiling. All processing is done until an EXE file is generated (~.EXE). When a warning is displayed, check the cause. Then, ignore the warning or modify the program as appropriate, and reexecute compilation. Warning displays can be cleared by using the [-w-] compiler option.

[Attention]

When an error is detected, the compiler skips data from the error occurrence position to the end of the statement. Therefore, a normal position immediately after an error is sometimes judged to be the error occurrence position. Or the position immediately after an error occurrence position sometimes cannot be detected.Therefore, the second error position and any subsequent ones may not always be correct.
The program line number and the ^ symbol which are displayed simultaneously with an error message indicate the location in the program the compiler read when the error was detected. The displayed location generally indicates the location near the error occurrence position. However, when an error is detected considerably after the compiler read the program, the displayed location indicates the location in the program considerably after the position of error occurrence. In order to solve this problem, if a program contains program lines which contain long complicated formulas and/or multiple statements which make an error difficult to detect, divide such formulas and lines into short statements.

(1) Fatal error

Error message	Meanings	Corrective Actions
can't create output file	An intermediate file $x.ASM (x is a numerical character) could not be created in the current directory.	The condition (directory full) in which a directory cannot be created is thought to be the cause. Delete any unnecessary files, and reexecute compilation.
can't create work file #1	An intermediate file BASIC$$$.INC could not be created in the current directory.	The condition (directory full) in which a directory cannot be created is thought to be the cause. Delete any unnecessary files, and reexecute compilation.
can't create work file #2	An intermediate file BC.TMP could not be created in the current directory.	The condition (directory full) in which a directory cannot be created is thought to be the cause. Delete any unnecessary files, and reexecute compilation.
compiler stack overflow	The compiler stack area overflowed.	Simplify any complex expressions. Reduce the number of nesting times: i.e., FOR - NEXT etc. (If the number of nesting times is 20 to 35, an error occurs.)
date area overflow(65000 bytes)	The data area necessary for variables and constants was too large to be secured	Since, in most cases, an error occurs because a huge array is declared, reduce the size of an array.
evaluation stack overflow	An expression in one statement was too complicated.	Reduce the complexity of the expression. For example, substitute the intermediate value of an expression for a variable.
file I/O error	An error occurred when accessing a source file or an intermediate file.	The disk is faulty or is not empty enough (disk is full). when the disk is full, delete any unnecessary files and reexecute compilation.
line too long	A source program line was too long. (If the line consists of 299 characters or more, an error occurs.)	Either it is not a source file stored by interpreter BASIC of the AD51H-S3 or a source file has been damaged. Store it again correctly.
source file 'XXXX' not found	The source file designated in the command line was not found.	Designate a correct source file name.
symbol table overflow	The number of variables, labels, or FN functions was excessive. (If there are 500 variables that consist of 9-character names, an error occurs.)	Change any variables, labels, or FN functions with long names into short ones. In addition, delete any unnecessary variables, labels, or FN functions.
too many target line numbers	There are a great many target line numbers referred to in statements such as GOTO and GOSUB.	Replace such parts with labels or divide the program.
unexpected end of file in 'XXXX"	A source file was interrupted in the middle of a statement.	Compile it after completing the program.
'dbb.obj' not found 'dbc.lib' not found	A start up module file or library file were not found.	The two dbb.obj and dbc.lib files must exist in a compiler start path or current directory. Confirm that these two files exist. Confirm that these two files exist.

Error message	Meanings	Corrective Actions
'XXXX' failed: error level X	An execution error was reported by the assembler or linker.	Use the assembler or linker manual to Investigate the cause of the error. [Attention] When executing a compiler by using Microsoft Macro Assembler Ver 4.0, if the [-4] option is not designated, this error occurs.
'XXXX' failed: Exec format error	The contents of the loading file of the assembler or linker were damaged.	Reregister the assembler or linker to the hard disk.
'XXXX' failed: No such file or directory	The assembler or linker were not found.	Place an assembler or linker in the current directory or the directory set in the environment variable PATH.
'XXXX' failed: Not enough memory	The assembler and a linker cannot be started because there is not enough memory.	Cancel the resident PRINT commands, etc. of the resident program. Remove any unnecessary device drivers. Or, make the designation of BUFFERS of CONFIG.SYS small. Taking these steps will increase the usable memory .

(2) Error messages

Error message	Meanings	Corrective actions
bad line number XXXXX	Incorrect syntax: There is a line number outside the range from 1 to 65529.	Designate the line number in the range from 1 to 65529.
DEF --- syntax error	Incorrect syntax: There is a syntax error in the DEFINT, DEFSNG and DEFDBL and DEFSTR commands.	Modify the program so that it has correct syntax.
DEF --- what ?	Incorrect syntax: There is a syntax error in the DEF command.	Modify the program so that it has correct syntax.
DIM --- syntax error	Incorrect syntax: There is a syntax error in the DIM command.	Modify the program so that it has correct syntax.
divide by 0	Improper parameter: There is a division by zero (/, \, MOD) in the numeric expression.	Modify the program so that it has no division by zero. [Attention] The compiler detects only the cases where a constant is divided by constant zero. When executing the program, division by zero is not detected as an error, and the maximum numerical value is returned. Single precision : +1.70141E+38 Double precision : +1.70141183460469D+38
expression too complex	Compiler restrictions: The numeric expression of a real number is too much complicated.	Reduce the complexity of the expression by taking some measures such as substituting an intermediate value of an expression for a variable.
FOR --- syntax error	Incorrect syntax: • The control variable and a substitution statement of an initial value are not found in the FOR~NEXT command. • A character-string variable is used as a control variable by mistake.	Designate a control variable or an initial value correctly.
FOR without NEXT	Incorrect syntax: • The NEXT command which corresponds to the FOR command is not found. • The FOR command and the NEXT command are not corresponding correctly.	Modify the program so that the FOR command and the NEXT command correspond correctly.
GOSUB not found GOSUB/GOTO not found GOTO not found	Incorrect syntax: The GOSUB command or the GOTO command is not found in the ON XX GOSUB command or the ON XX GOTO command.	Modify the program so that it has correct syntax.

Error message	Meanings	Corrective actions
illegal constant	Incorrect syntax: A character that is not recognized as a number is used in the octal constant and hexadecimal constant.	Modify the program so that it has correct syntax.
illegal parameter	Improper parameter: • There is an improper parameter. • There is a thing other than a variable or an arrangement name in the place where a variable or an arrangement name is needed.	Modify a program by using a proper parameter.
index must be 0...32766	Compiler restrictions: The value that designates the size of an arrangement in the DIM command is too large. Or, a variable or an expression is used to designate the size.	Reduce the size. Change the value that designates the size into a constant.
INPUT --- , or ; not found	Incorrect syntax: A thing other than ", " or ";" is found after the INPUT " ay string " command.	Modify the program so that it has correct syntax.
LINE INPUT --- ';' not found	Incorrect syntax: A thing other than " , " is found after the LINE INPUT "ay string" command.	Modify the program so that it has correct syntax.
LINE INPUT --- must be string variable	Incorrect syntax: A thing other than a character-string variable is designated to the string variable in the LINE INPUT command.	Designate a character-string variable to the string variable.
line number not found	Incorrect syntax: There is not a line number in the source program.	Describe a line number.
line number or label not found	Incorrect syntax: • There is not a line number or a label after the GOTO command or the GOSUB command. • An improper label name is used in the THEN, ELSE, RETURN, RESUME and RESTORE commands.	• Designate a line number or a label. • Designate a correct label name.
line number XXXXX not sequential	Incorrect syntax: The line number of a source program is not designated in ascending order.	Designate the line number in ascending order.
missing operand	Incorrect syntax: There is not an argument or an expression in the place where there must be an argument or an expression.	Designate an argument or an expression.
NEXT without FOR	Incorrect syntax, compiler restrictions: The NEXT command without a corresponding FOR command is found.	Describe a corresponding FOR command.

Error message	Meanings	Corrective actions
ON --- line number or label not found	Incorrect syntax: There is a syntax error in the row of line numbers or labels in the ON XX GOSUB command or the ON XX GOTO command.	Modify the program so that it has correct syntax.
ON --- string expression not allowed	Incorrect syntax: A character-string expression is designated in the ON <expression> GOSUB command and the ON <expression> GOTO command.	Designate a numeric expression to <expression>
OPEN --- file name not found	Incorrect syntax: There is not a character string of a file name in the OPEN command.	Designate a correct file name.
OPEN --- INPUT/OUTPUT/APPEND not found	Incorrect syntax: There is a thing other than the INPUT, OUTPUT and APPEND commands after the OPEN "XXX" FOR command.	Designate either INPUT, OUTPUT or APPEND command.
parameter must be numerical expression	Improper parameter: There is a thing other than a numeric expression in the place where a numeric expression is necessary.	Designate a numeric expression.
parameter must be string expression	Improper parameter: There is a thing other than a character string in the place where a character string is necessary.	Designate a character string.
parameter must be variable	Improper parameter: There is a thing other than a variable in the place where a variable is necessary.	Designate a variable.
port number must be constant	Improper parameter: The port number (%X) must be an integer constant.	Designate an integer constant.
PRINT USING --- ';' not found	Incorrect syntax: There is a character that is not ";" next to the format character string in the LPRINT and PRINT USING commands.	Change the character to ";".
PRINT USING --- format string not found	Incorrect syntax: There is a thing other than a format character string after the USING command in the LPRINT and the PRINT USING commands.	Designate a format character string.
PRINT USING --- illegal parameter	Incorrect syntax: There is a syntax error in the row of data to display in the LPRINT and the PRINT USING commands. (The number of data to display is restricted to 8 in the compiler.)	Modify the program so that it has correct syntax.
PRINT USING --- too many parameters	Compiler restrictions: The number of data to display is too large in the LPRINT and the PRINT USING commands. (The number of data to display is restricted to 8 in the compiler.)	Split one PRINT USING command to several PRINT USING commands.

Error message	Meanings	Corrective actions
RESTORE --- line number or label not found	Incorrect syntax: There is a thing other than a line number or a label after the RESTORE command.	Designate a line number or a label.
RESUME --- syntax error	Incorrect syntax: There is a syntax error in the RESUME command.	Modify the program so that it has correct syntax.
statement expected	Incorrect syntax: There is other thing such as a constant or a symbol, etc. than a command or a function at the beginning of a sentence.	Modify the program so that a command or a function is put at the beginning of a sentence.
STEP --- string expression not allowed	Incorrect syntax: The increment value specified by the STEP command is a character-string expression in the FOR~NEXT command.	Modify the program so that it has correct syntax.
string expression not allowed	Improper parameter: A character-string variable or a character-string expression is used.	This error occurs when a character-string variable or a character-string expression is designated by mistake to the place where a numerical value variable or a numerical expression must be designated. Modify the program so that it has correct syntax.
string expression too complex	Compiler restrictions: A character-string expression is too much complicated.	Reduce the complexity of the expression by taking some measures such as substituting an intermediate value of an expression for a variable,
subscript out of range	Improper parameter: • The value of the subscript of an array is outside the range specified in the DIM command. • Dimensionality specified in the DIMM command and the one actually written in the program are not the same.	• Modify the program so that the value of the subscript of an array is within the range specified in the DIM command. • Modify the program so that the dimensionality specified in the DIMM command and the one actually written in the program are the same.
swap type mismatch	Improper parameter: Types of two variables to be exchanged are not the same type.	Modify the program so that types of two variables to be exchanged are the same.
syntax error	Incorrect syntax: There is a syntax error in a command or a function.	Modify the program so that it has correct syntax.
syntax error --- binary operator	Incorrect syntax: There is only one term for a binary operator.	Modify the expression so that it is a correct expression.
syntax error at end of statement	Incorrect syntax: There is a reserved word, a symbol, or an expression, etc. in the place that should be the statement end.	Modify the program so that it has correct syntax.
syntax error in expression	Incorrect syntax: There is a syntax error in the expression.	Modify the program so that it has correct syntax.

Error message	Meanings	Corrective actions
syntax error in function parameter list	Incorrect syntax: There is a syntax error in the formal argument of the FN function in the DEF FN function.	Modify the program so that it has correct syntax.
syntax error in parameter	Incorrect syntax: There is a syntax error in the row of the arguments of a statement (a command or a function).	Modify the program so that it has correct syntax.
syntax error in string expression	Incorrect syntax: • There is a syntax error in the character string expression. • Operators that cannot be used for character strings are used.	Modify the program by using operators that can be used for character strings.
THEN/GOTO not found	Incorrect syntax: The THEN and GOTO commands are not found after the conditional expression in the IF command.	Modify the program by using the THEN and GOTO commands.
TO --- string expression not allowed	Incorrect syntax: The end value that is indicated after the TO command becomes a character-string expression in the FOR command.	Designate a numeric expression or a numerical value variable as the end value.
TO not found	Incorrect syntax: The TO command is not found in the FOR command.	Modify the program by using the TO command.
type mismatch	Improper parameter: • The parameter type is not proper. • A character string is given to a place where a numerical value is needed as the argument of a command or a function. Or a numerical value is given to a place where a character string is needed as the argument of a command or a function.	Modify the program so that the parameter type is proper.
WEND without WHILE	Incorrect syntax, compiler restrictions: The WEND command without a corresponding WHILE command is detected.	Modify the program so that the WHILE and the WEND commands correspond correctly.
WHILE without WEND	Incorrect syntax: the WHILE command without a corresponding WEND command is detected.	Modify the program so that the WHILE and the WEND commands correspond correctly.
XXXXX --- file number not found	Incorrect syntax: There is not a file number in the statement XXXX. Or there is a thing other than a numerical value in the position where there must be a file number.	Designate a file number.
XXXXX --- name too long	Incorrect syntax: Name XXXXX of a variable or a FN function is too long. (A maximum of 15 characters)	Shorten the name to 15 characters or less.

Error message	Meanings	Corrective actions
XXXXX --- ON/OFF/STOP not found	Incorrect syntax: The ON, OFF and STOP commands are not found in the statement XXXXX where they are required.	Designate ON, OFF, and STOP commands.
XXXX --- redimensioned array	Compiler restrictions: Array variable XXXXX has more than one definition.	Modify the program so that an array variable is not redefined.
XXXX --- string variable expected	Improper parameter: There is a thing other than a character-string variable in the place where there must be a character-string variable in the statement XXXXX.	Designate a character-string variable.
XXXXX --- undefined function	Incorrect syntax: An undefined FN function is called.	Define the function, or designate a defined function.
XXXXX --- undefined label	Incorrect syntax: Label XXXXX that does not exist is referred to in the GOTO and the GOSUB commands, etc..	Designate an existing label.
XXXXX --- undefined line number	Incorrect syntax: Line number XXXXX that does not exist is referred to in the GOTO and the GOSUB commands, etc..	Designate an existing line number.
XXXXX --- undefined variable	Compiler restrictions: An undefined variable is referred to.	———
XXXXX expected 'X' expected	Incorrect syntax: There are other things than XXXXX or "X" in the position where there must be XXXXX or 'X'.	Designate XXXXX or 'X'.
XXXXX not supported	Compiler restrictions: XXXXX contains a statement (a command or a function) that cannot be used by the compiler.	Modify the program so that it does not contain a command or a function that cannot be used by the compiler.

(3) WARNING

Error message	Meanings	Corrective actions
STOP assumed to be END	The STOP command is compiled being assumed as the END command.	———
XXXXX ignored	The command or the function XXXXX is ignored.	———

APPENDIX 5 METHOD FOR REQUIRING TRIGONOMETRIC FUNCTIONS THAT ARE NOT PREPARED IN THE AD51H-BASIC

Trigonometric functions that are not prepared in the AD51H-BASIC can be required by combining existing trigonometric functions.
Expressions to require each function are shown in the table below.

Functions to be required	Expression
Arc sign	ARCSIN(X)=ATN(X/SQR(-X*X+1))
Arc cosine	ARCCOS(X)=-ATN(X/SQR(-X*X+1))+1.5708
Arc secant	ARCSEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*1.5708
Arc cosecant	ARCCSC(X)=-ATN(1/SQR(X*X-1)+(SGN(X)-1)*1.5708
Arc cotangent	ARCCOT(X)=-ATN(X)+1.5708
Cosecant	CSC(X)=1/SIN(X)
Cotangent	COT(X)=1/TAN(X)
Secant	SEC(X)=1/COS(X)
Hyperbolic sign	SINH(X)=(EXP(X)-EXP(-X))/2
Hyperbolic cosine	COSH(X)=(EXP(X)+EXP(-X))/2
Hyperbolic tangent	TANH(X)=-EXP(-X)/(EXP(X)+EXP(-X))*2+1
Hyperbolic secant	SECH(H)=2/(EXP(X)+EXP(-X))
Hyperbolic cosecant	CSCH(X)=2/(EXP(X)-EXP(-X))
Hyperbolic cotangent	COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1
Hyperbolic arc sign	ARCSINH(X)=LOG(X+SQR(X*X+1))
Hyperbolic arc cosine	ARCCOSH(X)=LOG(X+SQR(X*X-1))
Hyperbolic arc tangent	ARCTANH(X)=LOG((1+X)/(1-X))/2
Hyperbolic arc secant	ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
Hyperbolic arc cosecant	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Hyperbolic arc tangent	ARCCOTH(X)=LOG((X+1)/(X-1))/2

Because there are cases that a certain measure of error arises, pay attention.

APPENDIX 6 RESERVED WORDS

Since the following reserved words all have specific meanings, they cannot be used for variables.

A : ABS
ACS
AND
ASC
ASN
ATN
AUTO

B : BEEP
BIN$
BLOAD
BSAVE

C : CALL
CDBI
CDBL
CHAIN
CHR$
CIDB
CINT
CISN
CLEAR
CLOSE
CLS
COLOR
COM
CONT
CONSOLE
COS
CSNG
CSNI
CSRLIN
CVB
CVD
CVDMBF
CVI
CVDMBF
CVS

D : DATA
DEF
DEFBYT
DEFDBL
DEFINT
DEFSNG
DEFSTR
DEF ZEVENT
DELETE
DIM
DRAW
DRD
DWT

E : ED
EDIT
ELSE
EM
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP

F : FIELD
FILES
FIX
FN
FOR
FORMAT
FRE

G : GET
GETMEN
GOSUB
GOTO
GO TO

H : HEX$

I : IF
IMP
INIT
INIT%
INKEY$
INP
INPUT
INPUT#
INPUT$
INPUT%
INSTR
INT

K : KEY
KILL

L : LEFT$
LEN
LET
LFILES
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET

M : MERGE
MID$
MKB$
MKD$
MKDMBF$
MKI$
MKS$
MKSMBF$
MOD
MON1
MON2

N : NAME
NEW
NEXT
NOT

O : OCT$
OFF
ON
OPEN
OR
OUT

P : PC
PEEK
PEEK2
POKE
POKE2
PORT
POS
PRCHE
PRD
PRESET
PRINT
PRINT#
PRINT%
PRRD
PRUN
PRWT
PSET
PUT
PUTMEM
PWT

R : RDSET
READ
REM
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN

S : SAVE
SEARCH
SGN
SIN
SPACE$
SPC
SPU$
SOR
STEP
STOP
STRING$
STR$
SWAP
SYSTEM

T : TAB
TAN
TC1
TC2
THEN
TIMER
TO
TROFF
TRON

U : USING
USR

V : VAL
VARPTR

W : WAIT
WTSET

X : XOR

Z : ZCLOSE
ZIDV
ZLDV
ZODV
ZOPEN
ZEVENT
ZMESSAGE
ZMESSAGE CLOSE
ZMESSAGE GET
ZMESSAGE KILL
ZMESSAGE OPEN
ZMESSAGE PUT
ZRELEASE
ZRESERVE
ZSIGNAL
ZSTART
ZURGENCY
ZWAIT DELAY
ZWAIT EVENT

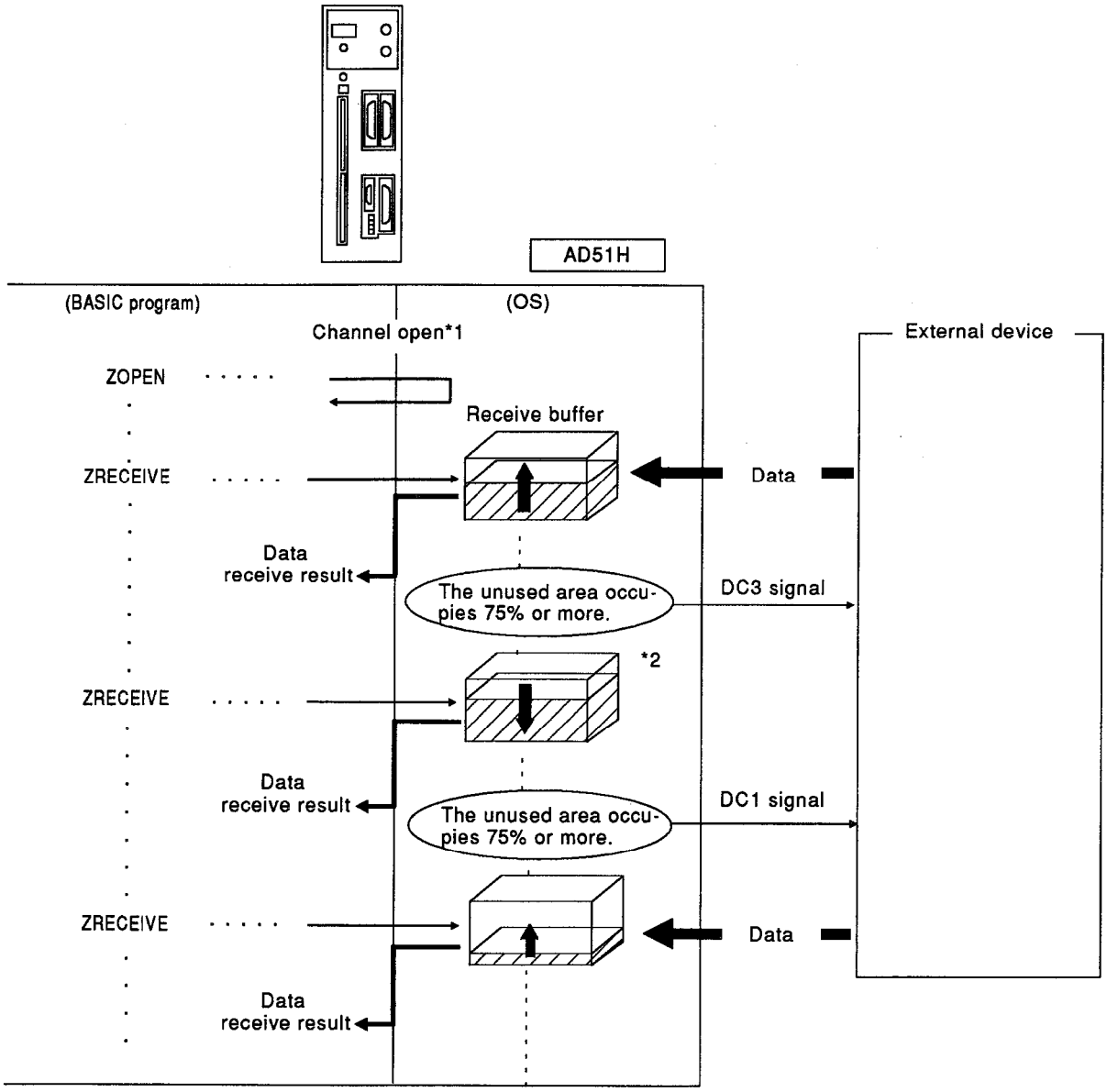
APPENDIX 7 COMMUNICATION CONTROLS

7.1 DC1/DC3 Controls

This section explains the data flow when DC1/DC3 controls are set to "enabled" in a communications channel.

7.1.1 DC1/DC3 send controls enabled

Whether a DC1 or DC3 signal is transmitted to an external device is determined by the size of the unused area of the designated receive buffer. Then, control of data transmitted from an external device to the AD51H is requested. After data received from the external device is stored, if the unused area occupies 25% or less of the receive buffer, a DC3 signal is transmitted to the external device.
After data received from the external device is stored, if the unused area occupies 75% or more of the receive buffer, a DC1 signal is transmitted to the external device.



*1 When a communications channel is opened, a DC1 signal is not transmitted to the external device. However, the communications channel state is the same as when a DC1 signal is transmitted.

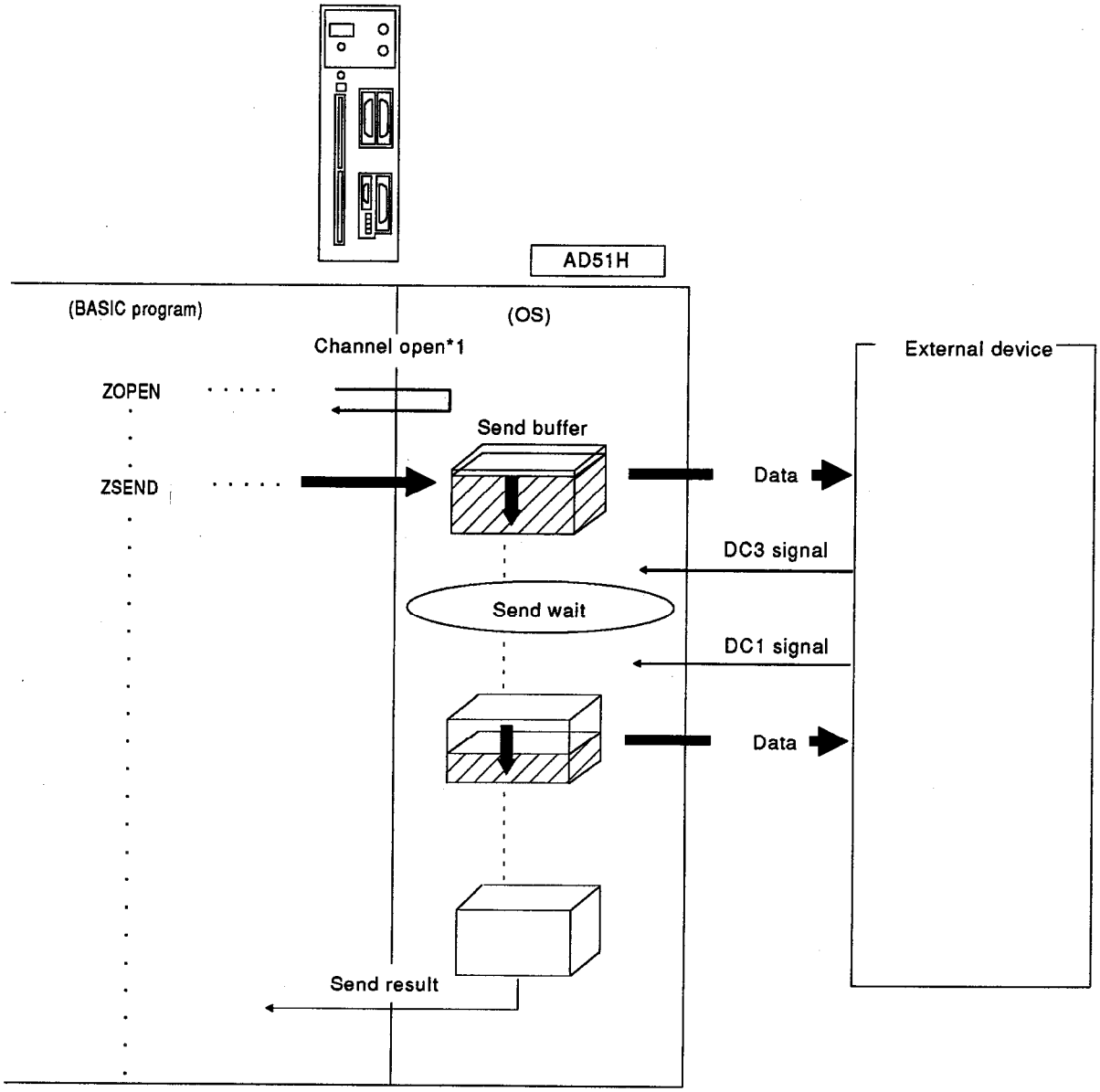
*2 After a DC3 signal is transmitted, received data is stored in the receive buffer until all unused areas of the receive buffer are filled with data.

POINT

To execute DC1/DC3 send control, specify a numerical value of 1 or more for the receive buffer when the communications channel is opened.

7.1.2 DC1/DC3 receive controls enabled

The data send from the BASIC program is controlled by the DC1 or DC3 signal received from an external device.
If a DC1 signal is received, the data send is started or continued.
If a DC3 signal is received, the data send is interrupted.
When a DC3 signal is received and the data send is interrupted, the rest of the interrupted data will be transmitted if a DC1 signal is received.



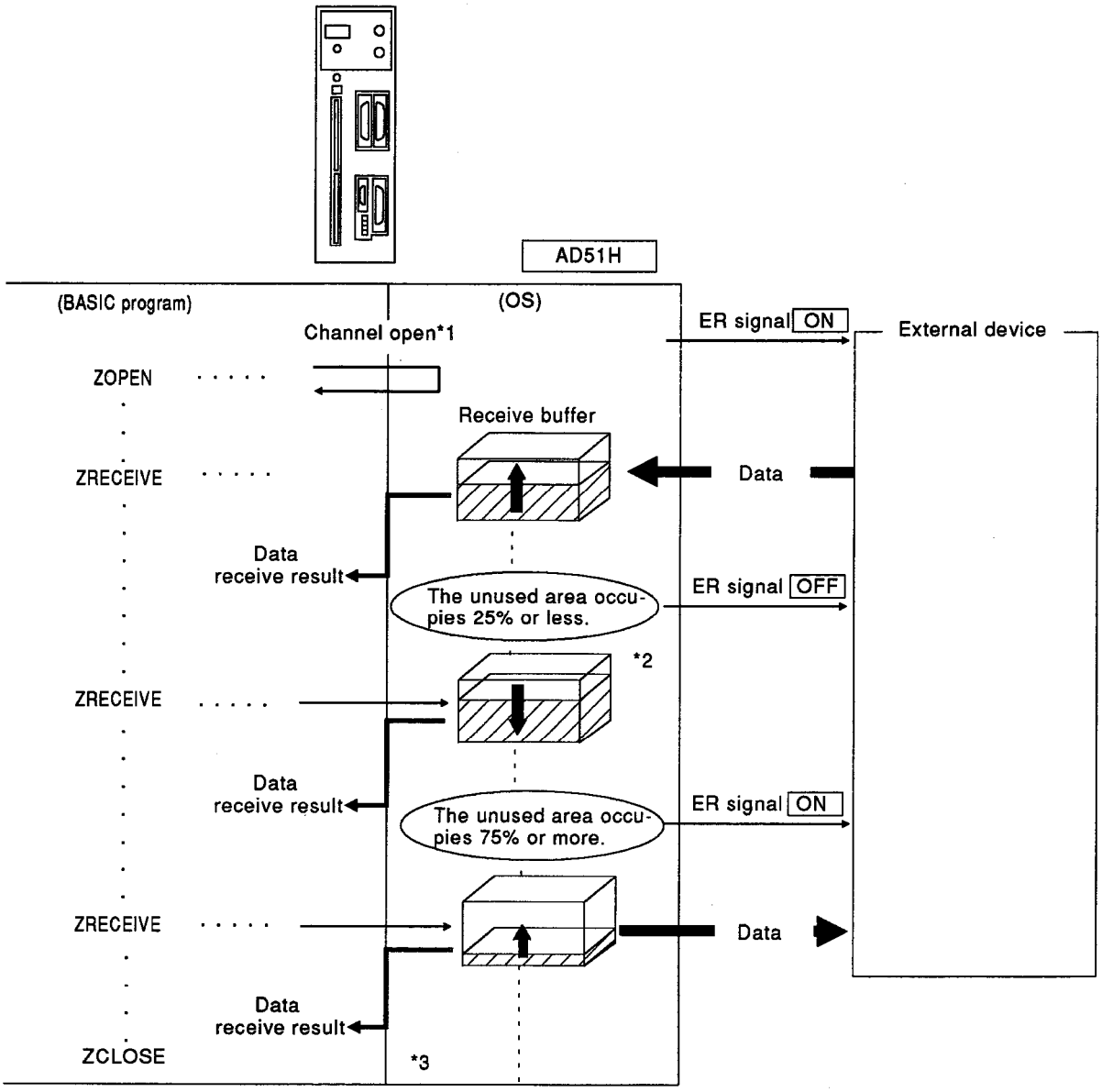
*1 When a communications channel is opened (even if a DC1 signal is not received from the external device), the state of the AD51H is the same as the state when a DC1 signal is received.

7.2 Signal Controls

This section explains the data flow when DC1/DC3 controls are set to "enabled" in the communications channel.

7.2.1 ER(DTR) control enabled

Whether the ER(DTR) signal of the RS-232C interface is turned ON or OFF is determined by the size of the unused area of the designated receive buffer. Then, control of data transmitted from an external device to the AD51H is requested.
After data received from an external device is stored, if the unused area occupies 25% or less of the receive buffer, the ER signal is turned OFF.
After data received from an external device is stored, if the unused area occupies 75% or more of the receive buffer, the ER signal is turned ON.



*1 When the AD51H starts up, the ER signal is OFF.
When a communications channel is opened, the ER signal goes ON.
After this, when data is received, the ER signal is turned ON or OFF according to changes in the size of the unused area of the receive buffer.

*2 Even if the ER signal is OFF, received data is stored in the receive buffer until all unused areas of the receive buffer are filled with data.

*3 Even if the communications channel is closed, the ER signal is not turned OFF.
The ER control is continued after the communications channel is closed and data received from the external device is stored in the receive buffer until all unused areas of the receive buffer are filled with data.
However, received data which was (a) left in the receive buffer when the communications channel was closed, or (b) stored in the receive buffer after the communications channel was closed cannot be read by the BASIC program.
When the communications channel is reopened, all received data in the receive buffer is cleared.

POINTS

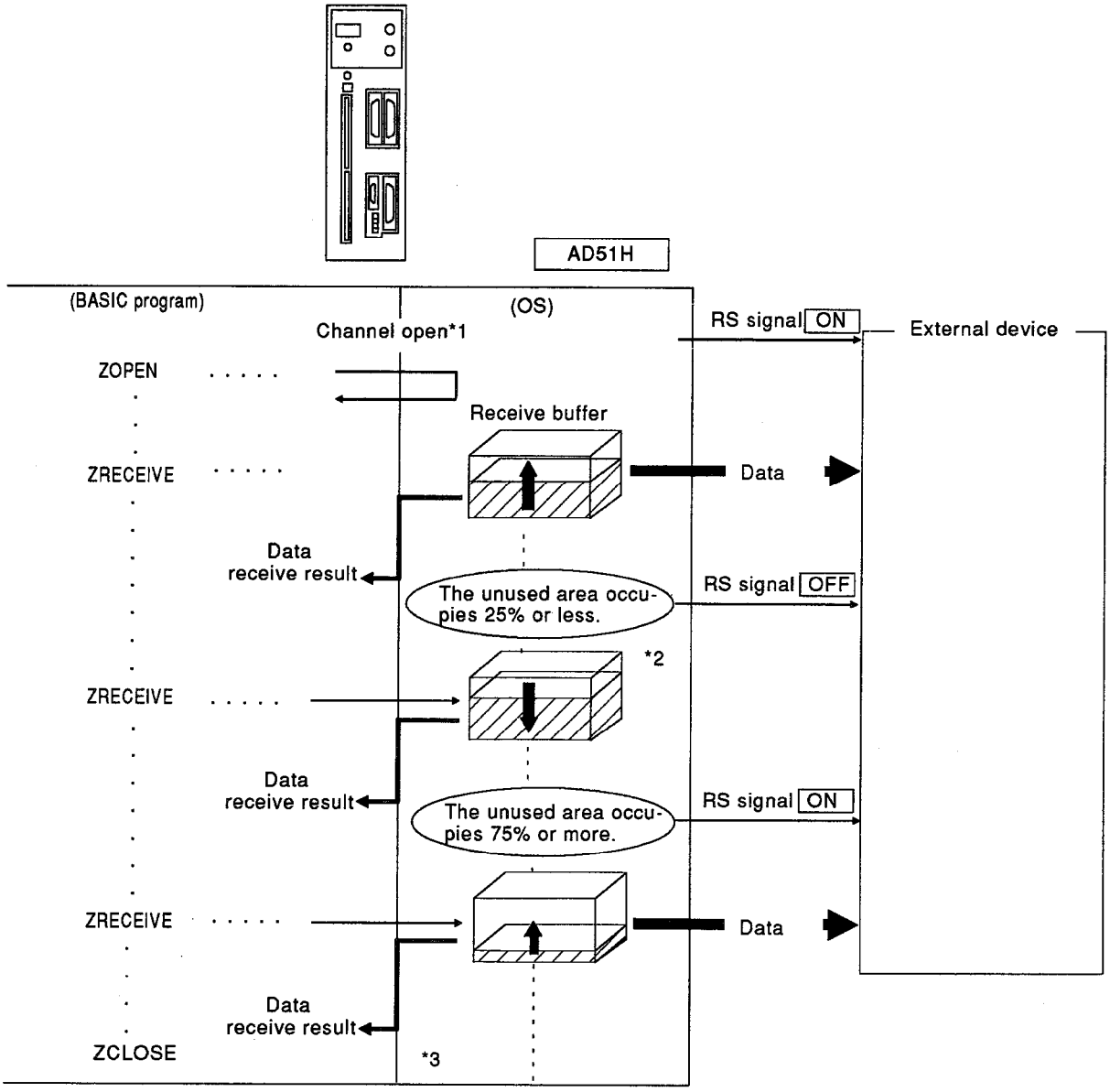
(1)To execute the ER(DTR) control, specify a numerical value of 1 or more for the receive buffer when the communications channel is opened.

(2)If the communications channel is opened without the ER control, the ER signal goes OFF.

(3)The RS(RTS) control can be used as a control function which is similar to the ER control.
Use AD51H control functions in accordance with the specifications of the external device.

7.2.2 RS(RTS) control enabled

Whether the RS(RTS) signal of the RS-232C interface is turned ON or OFF depends on the size of the unused area of the designated receive buffer. Then, control of data send from an external device to the AD51H is requested.
After data received from the external device is stored if the unused area occupies 25% or less of the receive buffer, the RS signal is turned OFF.
After data received from the external device is stored, if the unused area occupies 75% or more of the receive buffer, the RS signal is turned ON.



*1 When the AD51H starts up, the RS signal is OFF.
When the communications channel is opened, the RS signal goes ON.
After this, when data is received, the RS signal is turned ON or OFF according to changes in the size of the unused area of the receive buffer.

*2 Even if the RS signal is OFF, received data is stored in receive buffer until all unused areas of the receive buffer are filled with data.

*3 Even if the communications channel is closed, the RS signal is not turned OFF.
The RS control is continued after the communications channel is closed and data received from the external device is stored in the receive buffer until all unused areas of the receive buffer are filled with data.
However, received data which was (a) left in the receive buffer when the communications channel was closed, or (b) stored in the receive buffer after the communications channel was closed cannot be read by the BASIC program.
When the communications channel is reopened, all received data in the receive buffer is cleared.

POINTS

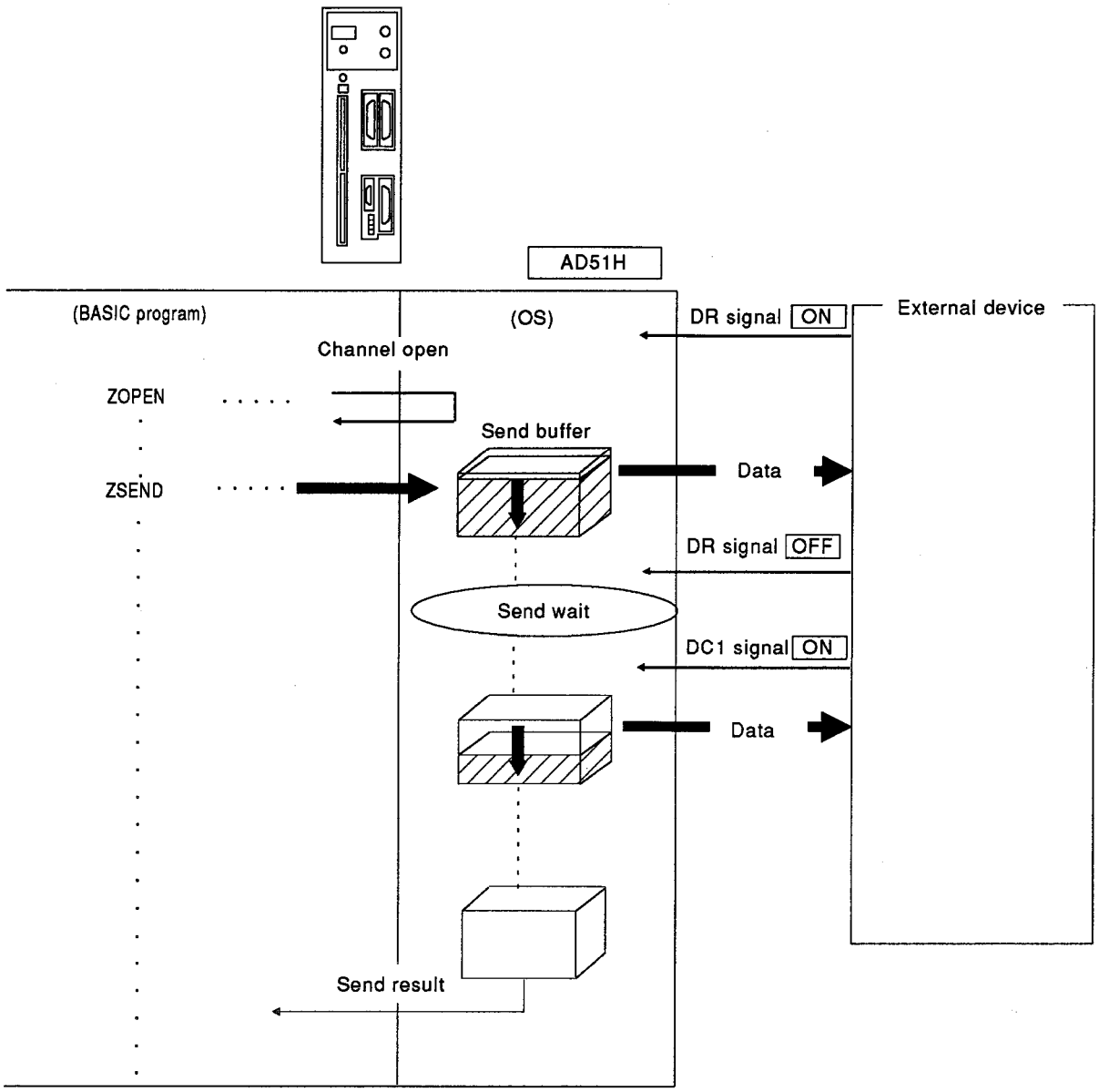
(1)To execute the RS(RTS) control, specify a numerical value of 1 or more for the receive buffer when the communications channel is opened.

(2)If the communications channel is opened without the RS control, the RS signal goes OFF.

(3)The RS(RTS) control can be used as a control function that is similar to the ER control.
Use AD51H control functions in accordance with the specifications of the external device.

7.2.3 DR(DSR) control enabled

Data send from the BASIC program is controlled by turning ON or OFF the DR(DSR) signal of the RS-232C interface of the AD51H.
When the DR signal is ON, data send is started or continued.
When the DR signal is OFF, data send is interrupted.
When the DR signal is turned OFF and data send is interrupted, the rest of the interrupted data will be transmitted if the DR signal is turned ON after that.

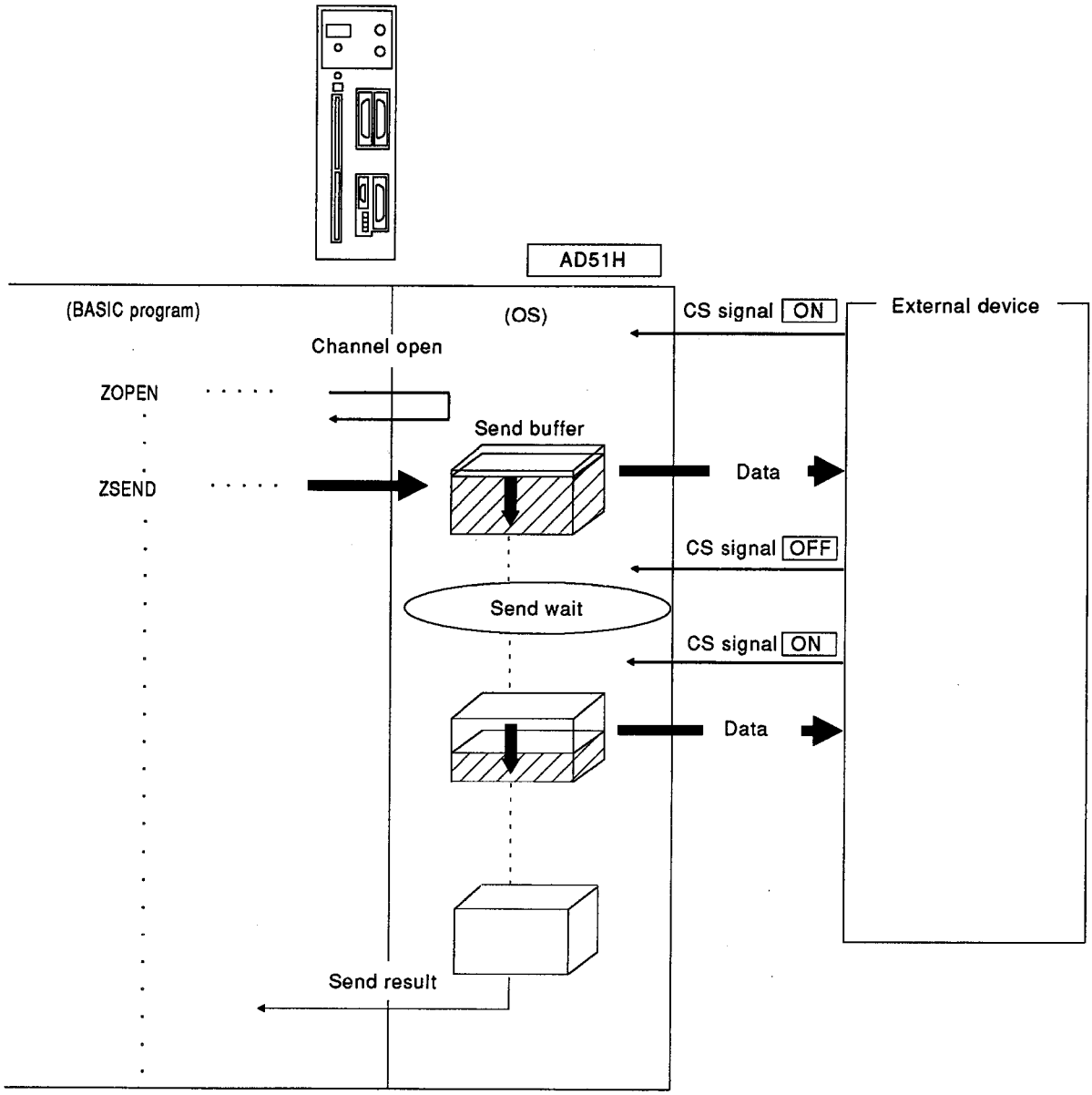


POINT

The CS(CTS) control can be used as a control function that is similar to the DR (DSR) control.
Use AD51H control functions in accordance with the specifications of the external device.

7.2.4 CS(CTS) control enabled

Data send from the BASIC program is controlled by turning ON or OFF the CS(CTS) signal of the RS-232C interface of the AD51H.
When the CS signal is ON, data send is started or continued.
When the CS signal is OFF, data send is interrupted.
When the CS signal is turned OFF and data send is interrupted, the rest of the interrupted data will be transmitted if the CS signal is turned on after that.



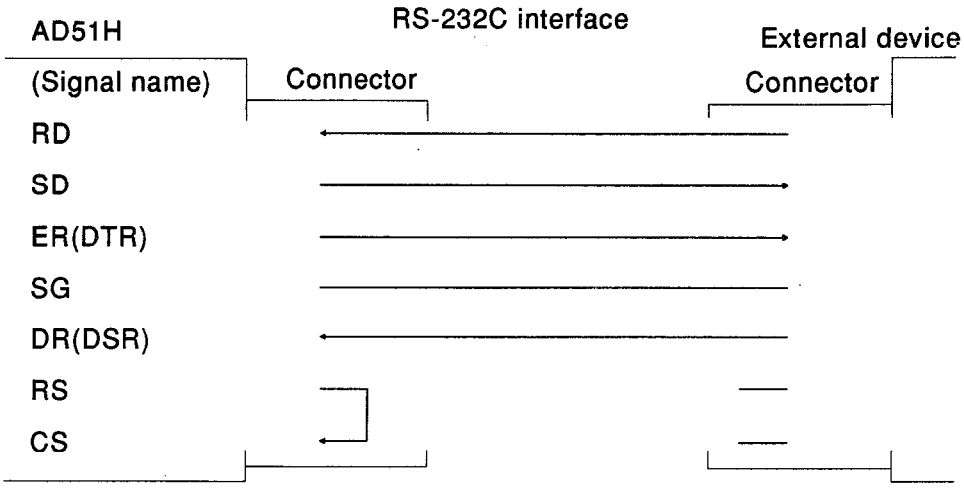
POINT

The DR(DSR) control can be used as a control function that is similar to the CS(CTS) control.
Use AD51H control functions in accordance with the specifications of the external device.

7.2.5 How to connect to an external device and precautions when designating signal controls

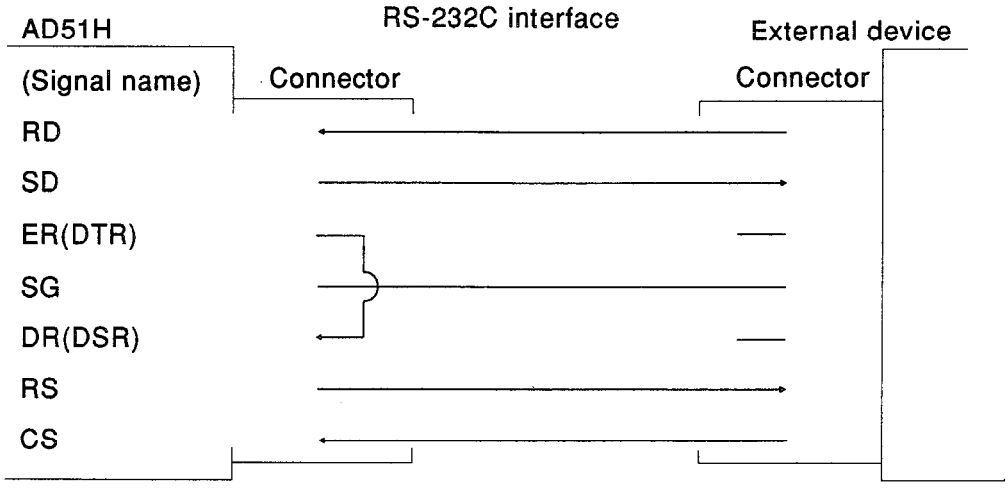
When connecting an AD51H to an external device, the following precautions when specifying signal controls should be taken.
An RS-232C interface is shown in this example.

(1) When the RS(TRS) signal line and the CS(CTS) signal line short-circuit on the AD51H side



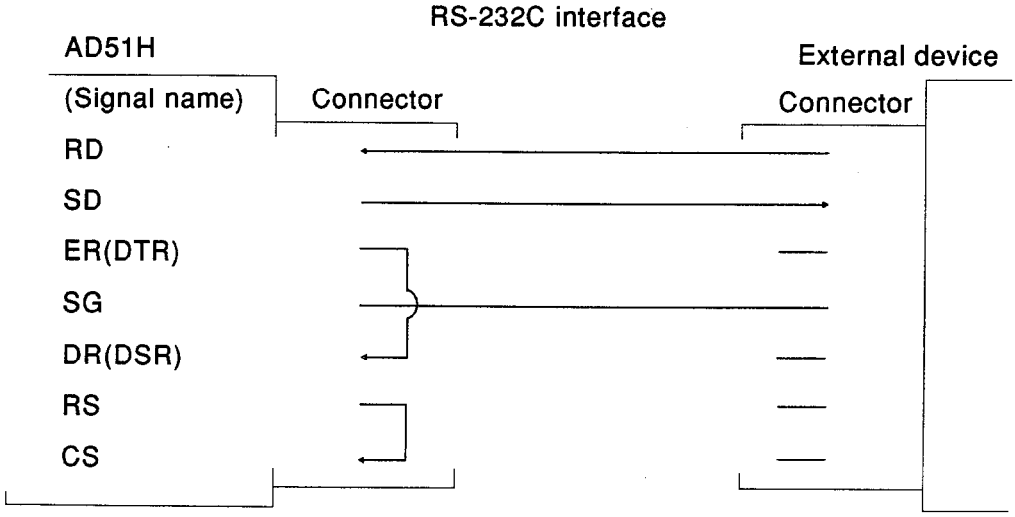
If the RS(RTS) control is set to "disabled", data from an external device can be received.
However, data cannot be transmitted to the external device.
To transmit data, set the RS control to "enabled" using processing code 32 of the ZCNTL command.

(2) When the ER(DTR) signal line and the DR(DSR) signal line short-circuit on the AD51H side



If the ER(DTR) control is set to "disabled" and the DR(DSR) control is set to "enabled", data from the external device can be received.
However, data cannot be transmitted to the external device.
To transmit data to the external device, set the DR controls to "disabled".

(3) When the RS(RTS) signal line, the CS(CTS) signal line, the ER(DTR) signal line, and the DR(DSR) signal line short-circuit on the AD51H side



If the RS(RTS) control or the ER(DTR) control is set to "disabled", data from the external device can be received.
However, data cannot be transmitted to the external device.
To transmit data to the external device, set the RS control to "enabled" using processing code 32 of the ZCNTL command, and set the DR(DSR) control to "disabled".

7.3 Break Character Control

When the BASIC program of an AD51H receives data from an external device, only the data up to the break character designated in the control table is stored in a variable area.
However, if a break character is not detected within the request number of characters designated by the receive request, received data equal in size to the request number of characters will be stored in a variable area.

A break character can be set using a request-to-receive ZRECEIVE command. Data that corresponds to a code from 00_H to FF_H can be designated as the break character according to the character-string length during the data send.

7.3.1 Example of break character specification

The following indicates the time when character data corresponding to the codes of 03_H and $0D_H$ is designated as the break character by the ZRECEIVE command, and received data is input. (It is assumed that the communication channel is already open.)

```
10  DIM TBL1%(2),TBL2%(4),TBL3%(2)
20  TBL1%(0)=9600
30  TBL1%(1)=&H108
40  TBL1%(2)=&H1
50  ZOPEN #2,TBL1%( )                  :OPENS THE CHANNEL
60  TBL2%(0)=22                        :SPECIFIES BREAK CHARACTER SETTING
70  TBL2%(1)=2                         :NUMBER OF BREAK CHARACTERS
                                        (TWO KINDS)
80  TBL2%(2)=&H300                     :BREAK CHARACTER
                                        (&H03 AND &H0D DATA)
90  ZCNTL #2,0,TBL2%( )                :EXECUTES SET BREAK CHARACTER
100  RD$=SPACE$(100)                   :STORES DUMMY
110  TBL3%(0)=100                      :REQUEST NUMBER OF BYTES (100 BYTES)
120  TBL3%(2)=200                      :TIME-OUT VALUE (20 SEC)
130  ZRECEIVE #2,0,TBL3%( ),RD$        :EXECUTES RECEIVE
```

In this case, the amount of received data (number of bytes) returned to the BASIC program returns to TBL% (1).
In addition, if the system program detects a break character, the detected break character is included in the final received data in RVBUF$.

Example

The Δ mark shows a space (20_H), and CR shows the carriage return code ($0D_H$).

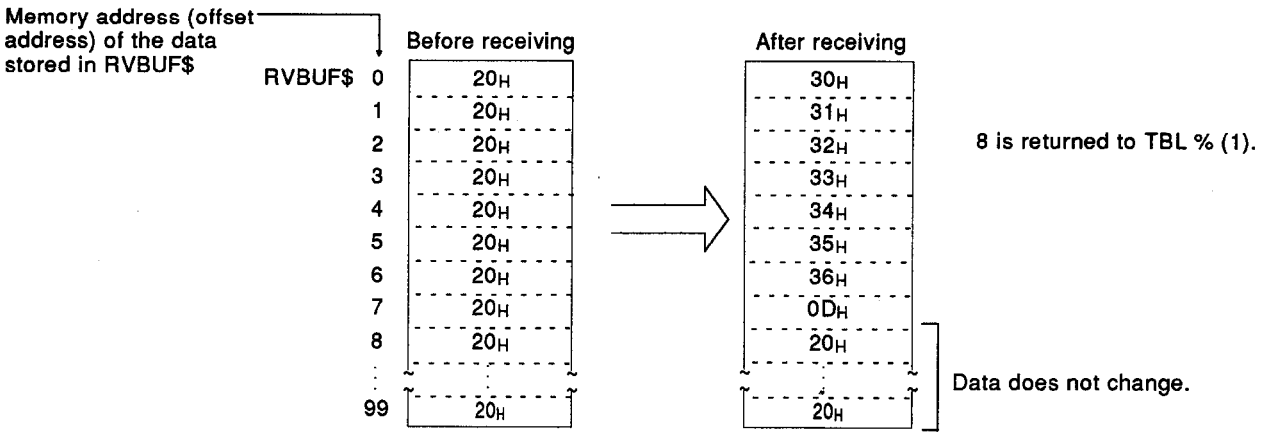
When the received data of "0123456$^{C}{}_{R}$" is returned

Description of RVBUF$ before executing

the ZRECEIVE command ..."ΔΔΔΔ.....ΔΔ"(100 bytes)

Description of RVBUF$ after executing

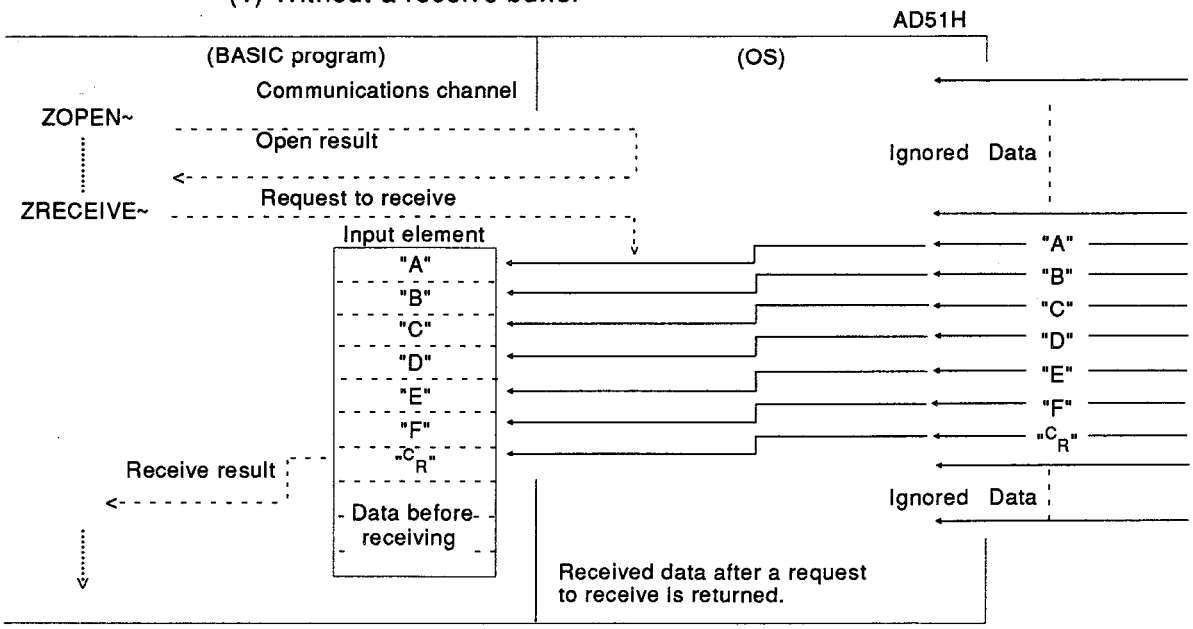
the ZRECEIVE command ..."0123456$^{C}{}_{R}$ΔΔΔ.....ΔΔ"(100 bytes)


Memory address (offset address) of the data stored in RVBUF$
Before receiving
After receiving
RVBUF$ 0
1
2
3
4
5
6
7
8
99
20H
30H
31H
32H
33H
34H
35H
36H
0DH
8 is returned to TBL % (1).
Data does not change.

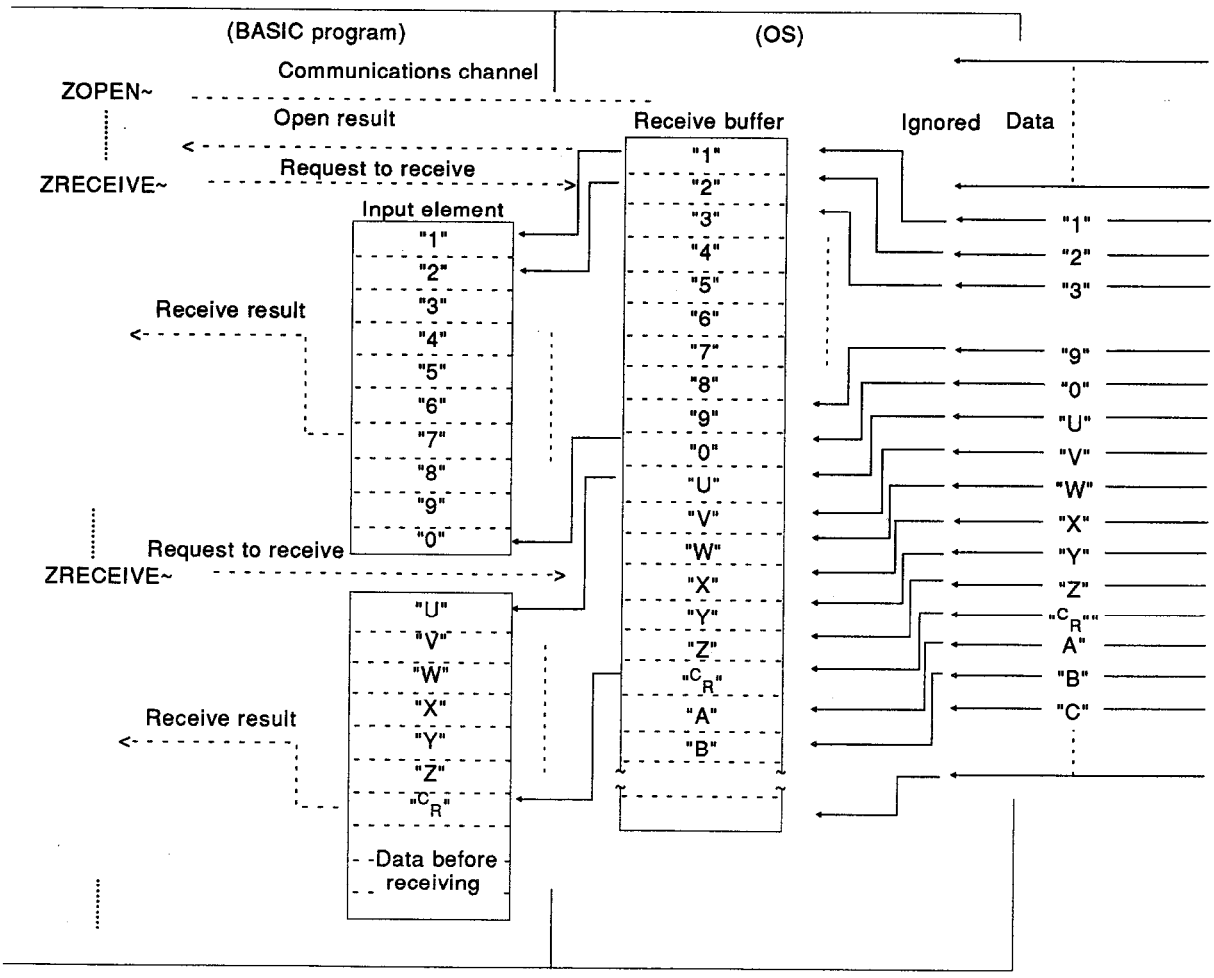
7.3.2 Received data flow when a break character is designated

When the communications channel is opened, the following example shows (a) when 0 (without a receive buffer) was designated for the receive buffer, and (b) when 1 to 1024 (with a receive buffer) was designated.
$0D_H(^{C}{}_{R})$ data is designated as the break character, and a request to receive 10 bytes of data (the maximum) is output.

(1) Without a receive buffer



(2) With a receive buffer



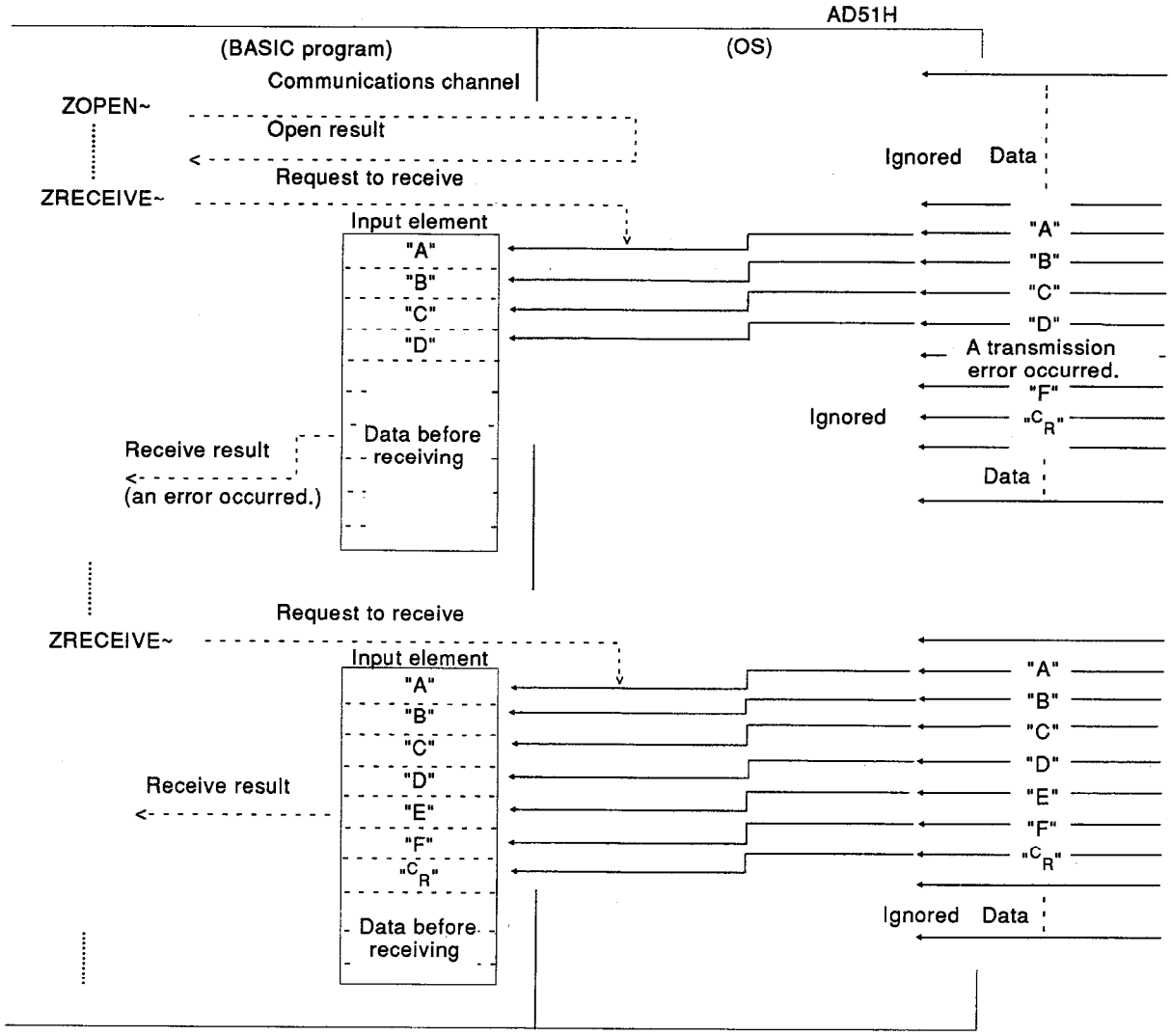
7.4 Data Flow When an Error Occurs During Data Receive

7.4.1 Data flow when a transmission error occurs

The following example shows the data flow when a transmission error (frame error, parity error, or overrun error) occurs when data is received from the external device.
0DH(CR) data is designated as a break character, and a request to receive of 10 bytes of data (the maximum) is output.

(1) Without a receive buffer

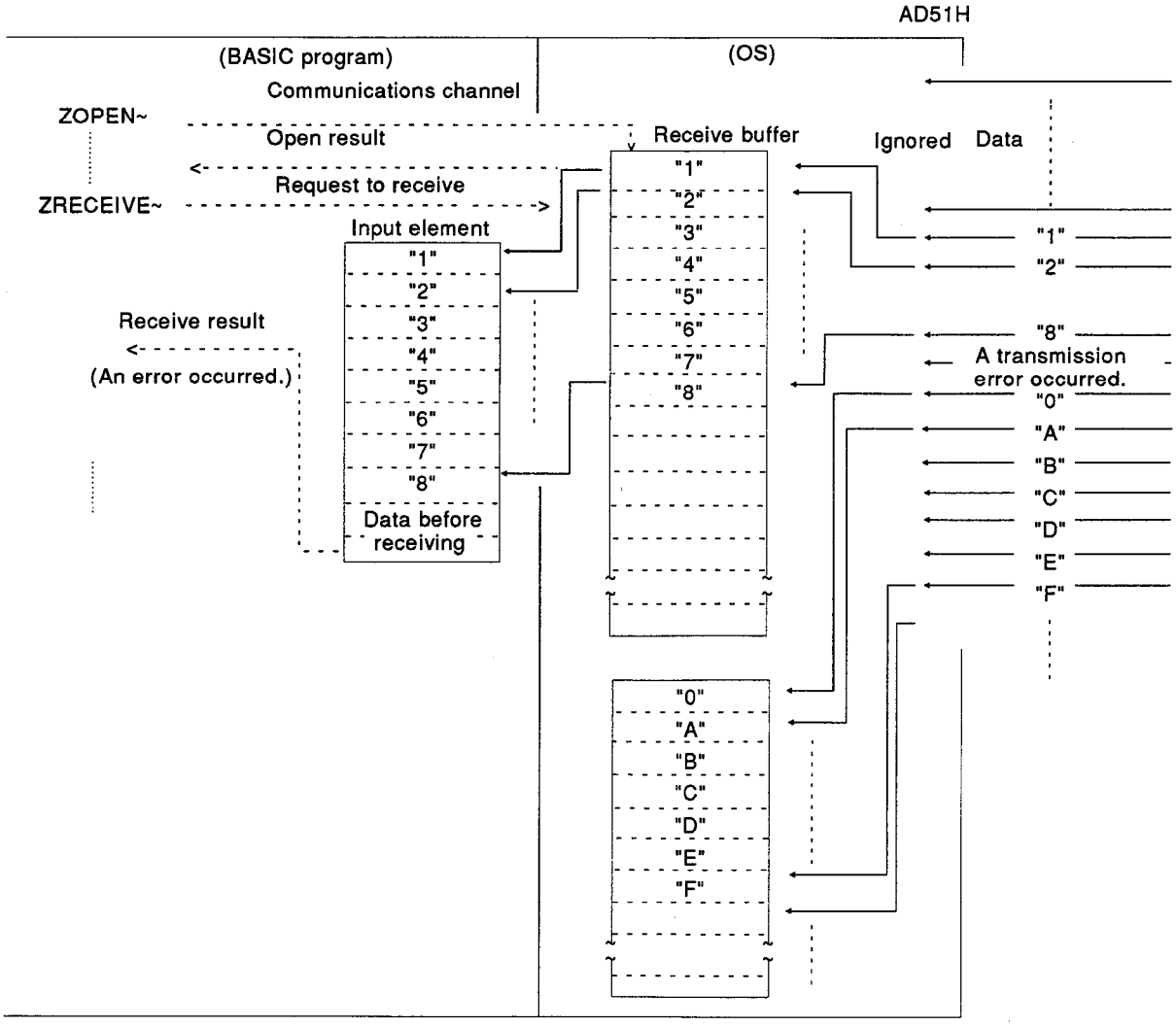
After getting a request to receive from the BASIC program, if a transmission error occurs during data receive from the host, any data received before the transmission error occurred is returned.



(2) With a receive buffer

(a) When data equal in size to the request number of characters is not stored in the receive buffer

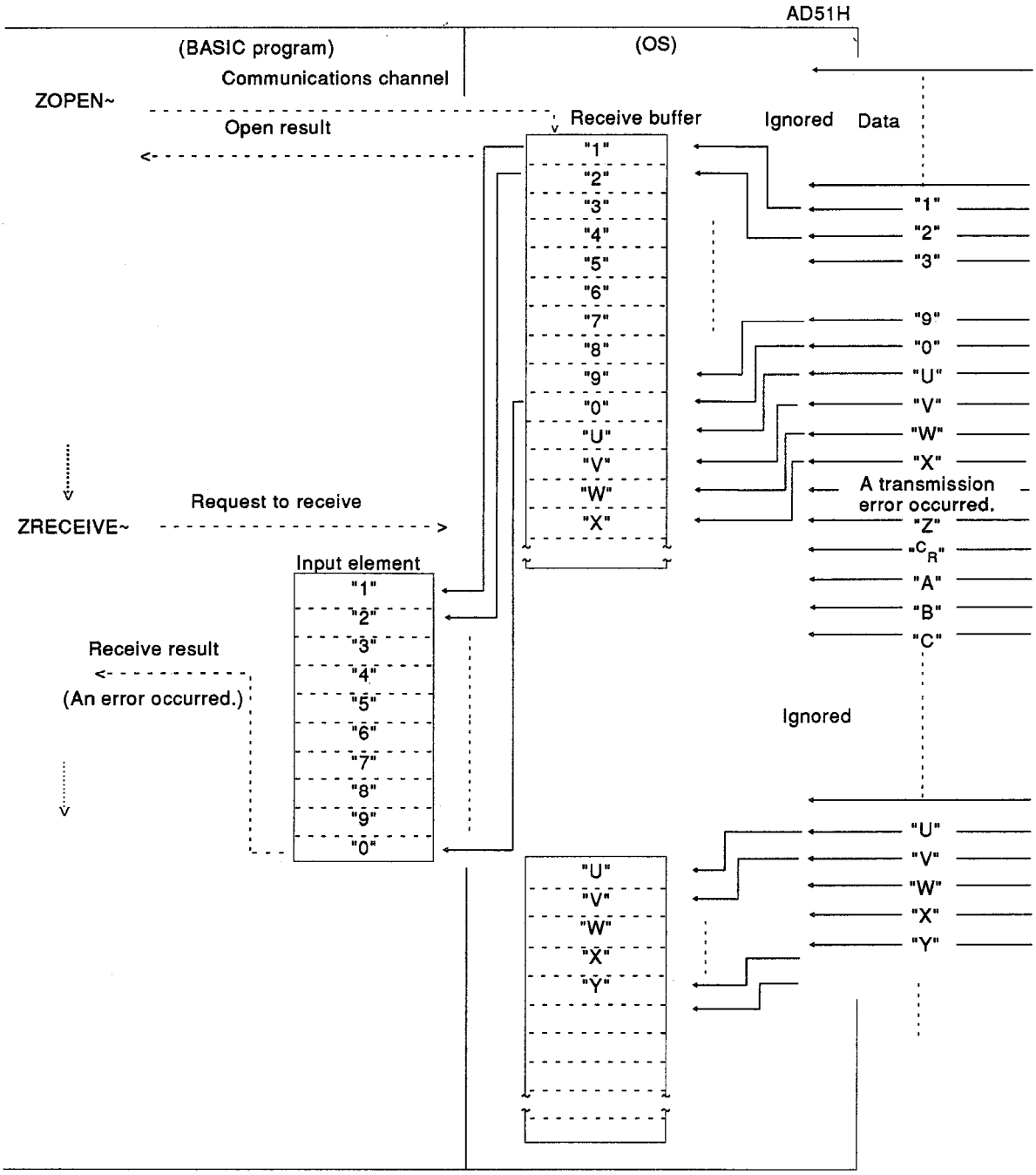
Received data stored in the receive buffer before a transmission error occurs is returned to the BASIC program.
Data received normally after a transmission error occurs is stored again in the receive buffer.
This data can be input using the BASIC program.



(b) When data larger than the request number of characters is stored in the receive buffer
Data received after a transmission error occurs is not stored in the receive buffer (it is ignored).
Data equal in size to the request number of characters is returned to the BASIC program.
If the request-to-receive processing has been completed, the following processing is executed.

1) All data in the receive buffer is cleared.

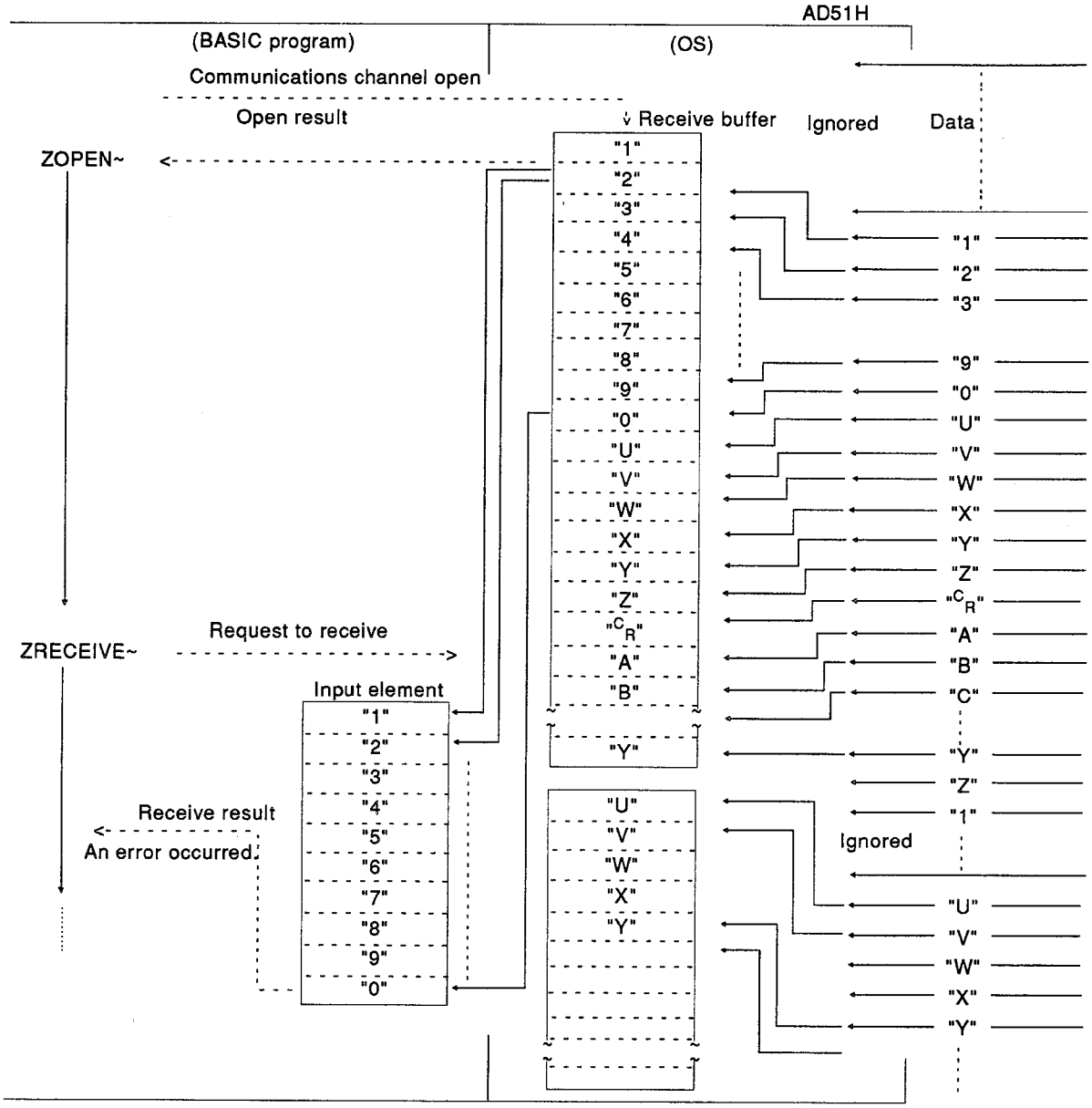
2) When processing of 1) has been completed, data received after that is stored in the receive buffer. (This received data can be input using the BASIC program.)



7.4.2 Data flow when a full receive buffer error occurs

This example shows the data flow under the following conditions:When the communications channel is opened, 1 to 1024 is then set for the receive buffer (with a receive buffer), and a full receive buffer error is caused by data received from an external device.
0DH($^{C}_{R}$) data is designated as the break character, and a request to receive 10 bytes of data (the maximum) is output.
Data received after a full receive buffer error occurs is ignored.
Data equal in size to the request number of characters is returned to the BASIC program.
If the request-to-receive processing has been completed, the following processing is executed.

1) All data in the receive buffer is cleared.
2) When processing of 1) has been completed, data received after that is stored in the receive buffer.
(This received data can be input using the BASIC program.)



7.5 How To Clear Received Data Stored in the Receive Buffer

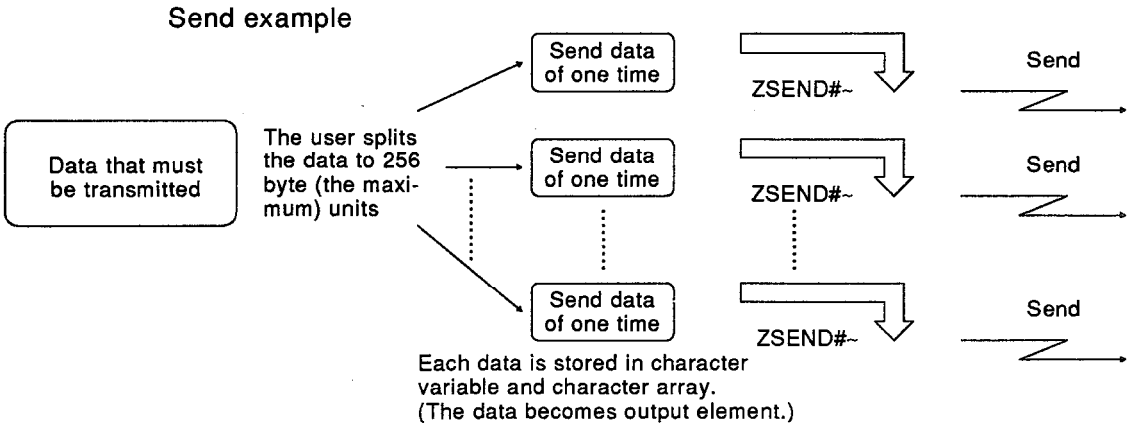
When the communications channel is opened, and when a numerical value of 1 or more is designated for the receive buffer, data received from an external device is stored to the receive buffer (which has a designated size).
Use either of the following methods to clear received data from the receive buffer and to store data received after that in the receive buffer.

1) Close the currently open communication channel, and then re-open it:
2) Set the request number of characters equal to the receive buffer size plus 1 for a time out value and output a request to receive using the ZRECEIVE command.
   All data returned by this request to receive will be ignored.
3) Changing a set value using the ZCNTL command.

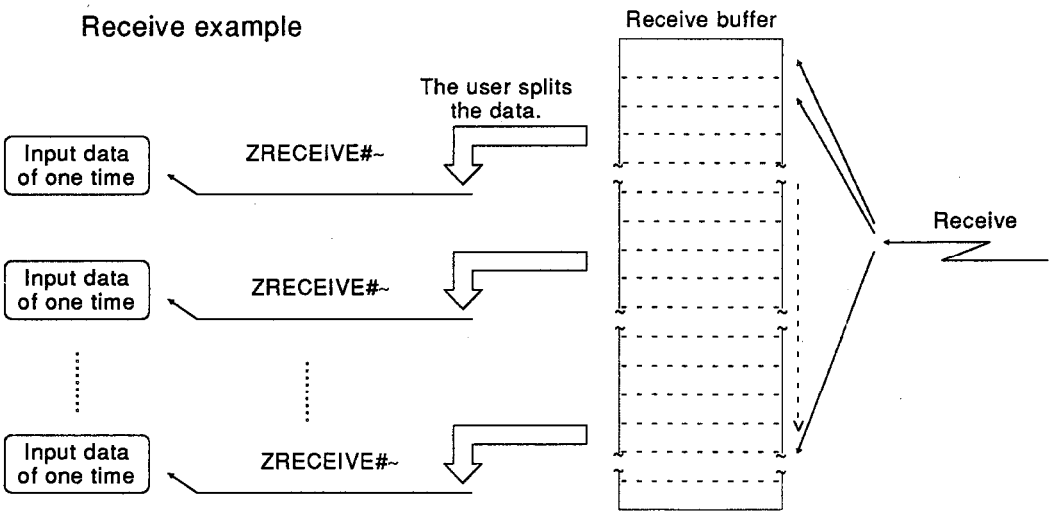
7.6 Communications Using 256 or More Bytes of Character Data

To transmit and receive 256 or more bytes of character data to/from an external device in the no-protocol mode through the RS-422 or RS-232C interface, split the data into 256 byte (the maximum) units.
This is because a one-time request to send or request to receive cannot process more than 256 bytes of character data using the BASIC program.

Send example



Receive example



IMPORTANT

Design the configuration of a system to provide an external protective or safety inter locking circuit for the PCs.

Under no circumstances will Mitsubishi Electric be liable or responsible for any consequential damage that may arise as a result of the installation or use of this equipment.

All examples and diagrams shown in this manual are intended only as an aid to understanding the text, not to guarantee operation. Mitsubishi Electric will accept no responsibility for actual use of the product based on these illustrative examples.

Owing to the very great variety in possible applications of this equipment, you must satisfy yourself as to its suitability for your specific application.

type AD51H-BASIC

Programming Manual

MODEL	AD51H-BASIC-P-E
MODEL CODE	13J754
IB(NA)66323-B(9302)MEE	

MITSUBISHI ELECTRIC CORPORATION

HEAD OFFICE : MITSUBISHI DENKI BLDG MARUNOUCHI TOKYO 100 TELEX : J24532 CABLE MELCO TOKYO
NAGOYA WORKS : 1-14 , YADA-MINAMI 5 , HIGASHI-KU, NAGOYA , JAPAN

When exported from Japan, this manual does not require application to the Ministry of International Trade and Industry for service transaction permission.

Specifications subject to change without notice.