

GPC-BASIC

HAND BOOK

MITSUBISHI PROGRAMMABLE CONTROLLER

MELSEC



MITSUBISHI

INTRODUCTION

This GPC-BASIC Grammar Book has been prepared to allow easy use of GPC-BASIC which is a programming language of the multi-purpose controllers KGPC1 and KGPC11 and the intelligent communication unit KD51 of Mitsubishi general-purpose programmable controller MELSEC-K series.

GPC-BASIC, which was developed in 1982 when the KGPC1 was released, is also used for the KGPC11 and KD51 and has fully established its position as a sublanguage of the MELSEC-K series. However, it is not easy to learn all of commands and command functions unique to GPC-BASIC, and it is troublesome to look up a command and function in the Instruction Manual and Programming Manual. For this purpose, therefore, it is recommended to keep this dictionary at hand and utilize it as required.

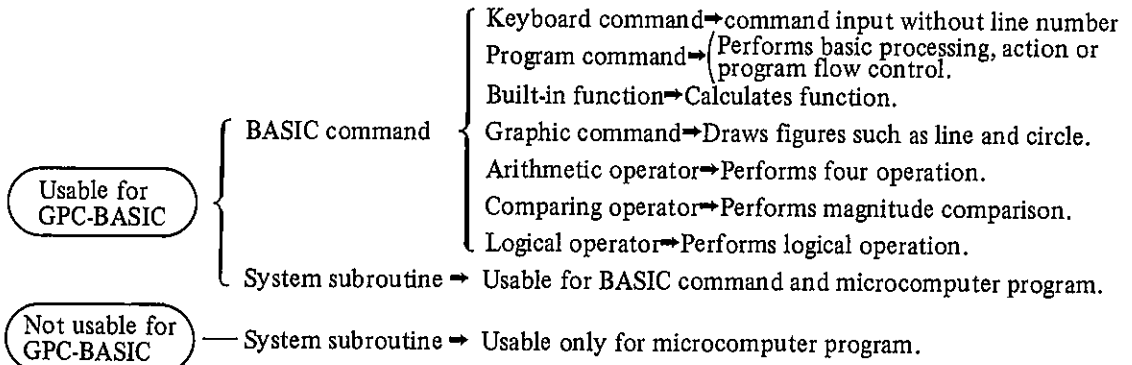
Although this dictionary has been published with the release of KD51, it is issued as an aid to better understanding of GPC-BASIC. Therefore, when you use the KGPC and KD51, please read the Instruction Manuals for these units.

March, 1985

MITSUBISHI ELECTRIC

Command List

Commands and system subroutines, which can be used for GPC-BASIC, are largely classified as shown below:

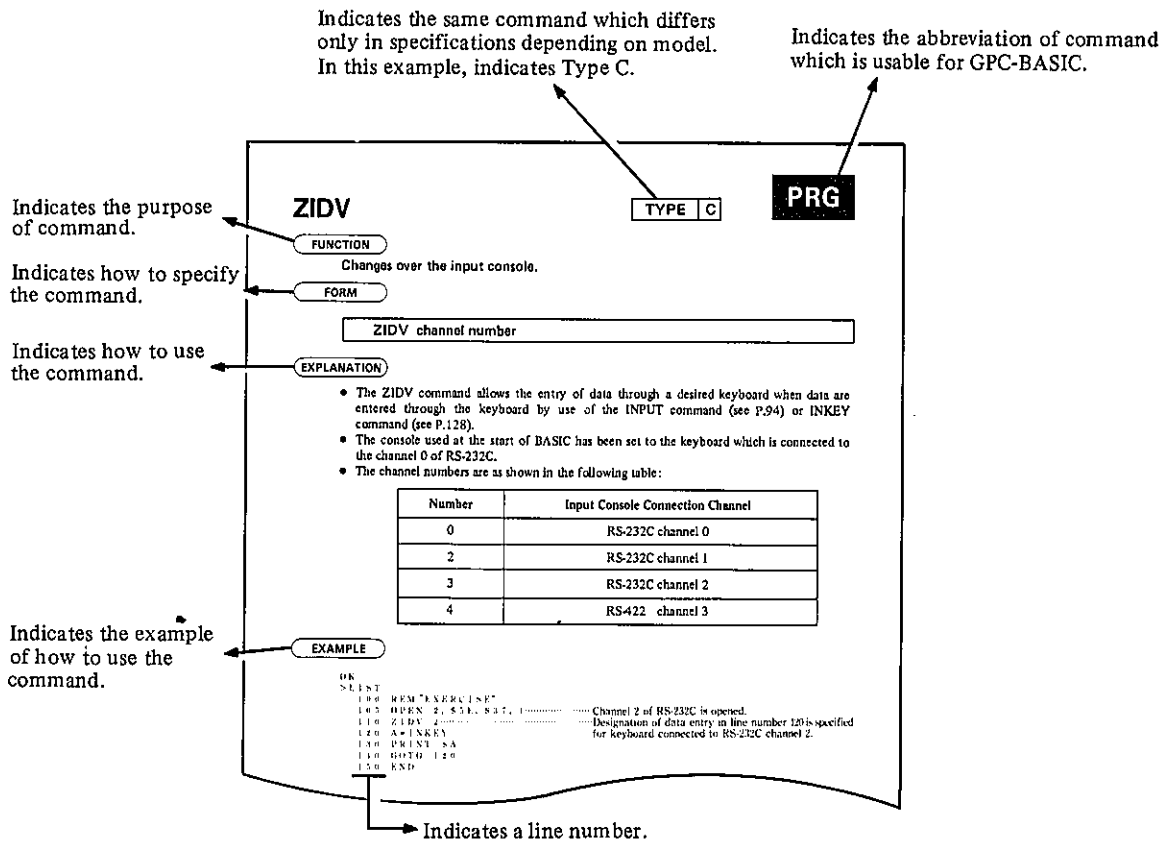


System subroutine is a subroutine which can be used by the user, among the subroutines unique to the system.

In the following pages, the BASIC commands are abbreviated as shown below:

Keyboard commandKEY
 Program commandPRG
 Built-in functionINT
 Graphic commandGRF
 Arithmetic operatorALU
 Comparing operatorCOM
 Logical operatorLOG

Each command is explained in the following format:



NOTE

Symbols shown in "FORM" have the following meanings:

- Enter the capitals in the indicated form.
- The portion in brackets [] can be omitted.
- When a portion is enclosed by parentheses () or double quotes " ", specify the portion with these symbols.
- The portion with repetition symbol can be specified repeatedly within the range of one line.

Command Name		Function	Type	Applicable Model			Reference page		
				GPC1	GPC11	KD51			
KEY	1	AUTO	Automatic generation of line number	-	•	•	•	57	
	2	BYE	Return to BASIC programming data display screen	-	•	•	•	58	
	3	CONT	Resumption of program run after BREAK	-	•	•	•	59	
	4	COMPILE	Compilation to multi task executable program	-		•	•	60	
	5	DELETE	Deletion of program from specified line number to specified line number	-		•	•	61	
	6	EDIT	Correction of statement in one line	-		•	•	62	
	7	EXECUTE	Run of program after "RUN" or "COMPILE"	-		•	•	65	
	8	LIST	Display of program on screen	A	•				66
				B		•			67
				C			•		68
	9	LLIST	Print-out of program	-	•	•	•	69	
	10	NEW	Deletion of program	-	•	•	•	70	
	11	RENUM	Renumbering of line numbers	A	•				71
				B		•	•		72
12	RUN	Run of program	-	•	•	•	73		
13	ZDV	Display of I/O console	-	•		•	74		
14	—	Deletion of line	-	•	•	•	75		
PRG	1	BREAK	Resumption of program run after run, temporary stop or "CONT"	-	•	•	•	78	
	2	CALL	Calling of machine language program	-	•	•	•	127	
	3	CLS	Erasure of CRT screen	A	•		•	81	
				B		•		82	
	4	CLOSE	Closing of specified channel of K30RSF or RS-232C of K31PST	A	•			83	
				B		•		84	
				C			•	85	
	5	COLOR	Designation of character color	-	•	•		79	
	6	END	End of program run	-	•	•	•	86	
	7	FOR...NEXT	Repeated run of program from "FOR" to "NEXT"	-	•	•	•	87	
	8	GOTO	Move to specified line number	-	•	•	•	89	
9	GOSUB...	Move to specified subroutine	-	•	•	•	90		
	...RETURN	Return from subroutine	-	•	•	•	90		
10	IF	Judgement of result of expression	-	•	•	•	91		
11	INPUT	Input through keyboard	A	•				93	
			B		•	•		94	

Table 1 List of BASIC commands

Command Name		Function	Type	Applicable Model			Reference page	
				GPC1	GPC11	KD51		
PRG	12	INKEY	Assignment of input through keyboard to variable	-	•	•	•	128
	13	LCOPY	Print-out of CRT screen	A	•			96
				B		•		97
	14	LET	Assignment of value of expression to variable	-	•	•	•	98
	15	LOCATE	Moving of cursor position	A	•			99
				B		•		100
				C			•	102
	16	LPRINT	Print-out of data	A	•			103
				B		•		104
				C			•	105
	17	ONGOSUB	Move to subroutine in line number specified by value of expression	-	•	•	•	106
	18	ON GOTO	Move to line number specified by value of expression	-	•	•	•	107
	19	OPEN	Opening of specified channel of K30RSF	A	•			108
			Opening of specified RS-232C channel of K30RSF or K31PST	B		•		111
			Opening of specified channel of RS-232C or RS-422	C			•	114
	20	PEEK	Read of one-byte data from specified address of memory	-	•	•	•	129
	21	POKE*	Write of one-byte data to specified address of memory	-	•	•	•	117
	22	PRINT	Display of data on screen	A	•			118
				B		•	•	121
	23	REM	Used to write comment. No influence is given to execution.	-	•	•	•	124
	24	SIZE	Display of text program capacity	A	•	•		130
				B			•	132
	25	STOP	Stop of program run	-	•	•	•	125
	26	ZCOFF	Erasure of cursor	A	•	•		136
			Underline start	B			•	137
	27	ZCON	Display of cursor (used after "ZCOFF")	A	•	•		138
			Underline stop (used after "ZCOFF")	B			•	139
	28	ZCRV	Reversion of CRT screen color and character color	-	•	•	•	140
	29	ZDATE	Display of year, month, day, hour and minute	-	•	•	•	141
	30	ZDSP	Change-over of CRT display mode	A	•			153
				B		•		155
31	ZIDV	Change-over of device of input console	A	•			142	
			B		•		143	
			C			•	145	

Table 1 List of BASIC Commands (Continued)

Command Name	Function	Type	Applicable Model			Reference page	
			GPC1	GPC11	KDS1		
PRG	32 ZMOV	Data transfer from memory to memory	-	•	•	•	157
	33 ZNOR	Restoration of reversed character and figure color after "ZCRV"	-	•	•	•	146
	34 ZODV	Change-over of device of output console	A		•		147
			B			•	149
	35 ZRD1	Read of one-byte data from specified channel	-	•	•	•	159
	36 ZRD2	Read of two-byte data from specified channel	-	•	•	•	161
	37 ZTIME	Interruption of execution for specified interval of time	-	•	•	•	151
	38 ZWR1	Write of one-byte data to specified channel	-	•	•	•	163
	39 ZER2	Write of two-byte data to specified channel	-	•	•	•	165
INT	1 ABF	Absolute value of value (real number) of mathematical expression	-	•	•	•	168
	2 ABS	Absolute value of value (integer) of mathematical expression	-	•	•	•	169
	3 ACOS	Arccosine (\cos^{-1}) of mathematical expression	-	•	•	•	170
	4 ASIN	Arcsine (\sin^{-1}) of mathematical expression	-	•	•	•	171
	5 ATAN	Arctangent (\tan^{-1}) of mathematical expression	-	•	•	•	172
	6 COS	Cosine (cos) of mathematical expression	-	•	•	•	173
	7 EXP	Value of exponential function of which base is "e" (e=2.718281)	-	•	•	•	174
	8 LN	Value of natural logarithm (loge)	-	•	•	•	175
	9 LOG	Value of common logarithm (\log_{10})	-	•	•	•	176
	10 NOT	Generation of "1" when value of mathematical expression is "0" and generation of "0" when the value is other than "0".	-	•	•	•	177
	11 RND	Assignment of random number to variable	-	•	•	•	178
	12 SIN	Sine (sin) of mathematical expression	-	•	•	•	179
	13 SQRT	Value of square root of mathematical expression	-	•	•	•	180
	14 TAN	Tangent (tan) of mathematical expression	-	•	•	•	181
GRF	1 CIRCLE	Drawing of circle and sector	-	•			183
	2 FSTYLE	Designation of details of "FSTYLE"	-		•		185
	3 FTYPE	Painting of specified area	-		•		187
	4 GCOLOR	Designation of color for graphic command	-		•		189
	5 LINE	Drawing of straight line	-		•		191
	6 LTYPE	Designation of straight line type	-		•		193
	7 MODE	Logical operation of figure color	-		•		195
	8 ORIGIN	Moving of figure to specified position	-		•		201
	9 PRESET	Erasure of point in specified dot coordinates	-		•		204
	10 PSET	Drawing of point in specified dot coordinates	-		•		207

Table 1 List of BASIC Commands (Continued)

Command Name		Function	Type	Applicable Model			Reference page
				GPC1	GPC11	KD51	
GRF	11	VIEW	Designation of display area	—	•		208
	12	WINDOW	Allotment of coordinates of display area	—	•		211
ALU	1	+	Addition	—	•	•	30
	2	—	Subtraction	—	•	•	30
	3	*	Multiplication	—	•	•	30
	4	/	Division	—	•	•	30
	5	^	Exponent	—	•	•	30
	6	—	Sign reversion	—	•	•	30
	7	%	Remainder calculation	—	•	•	31
COM	1	=	Is equal to	—	•	•	32
	2	#	Is not equal to	—	•	•	32
	3	<	Is less than	—	•	•	32
	4	>	Is greater than	—	•	•	32
	5	<=	Is not greater than	—	•	•	32
	6	>=	Is not less than	—	•	•	32
LOG	1	#	Negation (NOT)	—	•	•	33
	2	&	Logical product (AND)	—	•	•	33
	3	!	Logical sum (OR)	—	•	•	33
	4	\	Exclusive logical sum (EXOR)	—	•	•	33

Table 1 List of BASIC Commands (Continued)

Note 1: — in the Type column indicates the command which is commonly usable for each model.

Note 2: The command having A, B and C in the Type column differs in specifications depending on model.

Note 3: • indicates the command which is usable for the model.

Note 4: ALU COM LOG commands are explained in Section 2.5. (They are not explained in Section 3 "BASIC COMMANDS".)

	System Subroutine	Function	Type	Applicable Model			Reference Page
				GPC1	GPC11	KD51	
1	SAI	ASCII (hexadecimal) → BIN	-		•	•	223
2	SIA	BIN → ASCII (hexadecimal)	-		•	•	225
3	SAN	ASCII (decimal) → BIN	-		•	•	227
4	SNA	BIN → ASCII (decimal)	-		•	•	229
5	SAF	ASCII → real number	-		•	•	231
6	SFA	Real number → ASCII	-		•	•	234
7	SBF	Integer → real number	-		•	•	236
8	SFB	Real number → integer	-		•	•	237
9	SBD4	BIN → BCD (4 digits)	-		•	•	239
10	SDB4	BCD → BIN (4 digits)	-		•	•	240
11	SBD6	BIN → BCD (6 digits)	-		•	•	241
12	SDB6	BCD → BIN (6 digits)	-		•	•	243
13	SBA	BIN addition (24 bits)	-		•	•	245
14	SBS	BIN subtraction (24 bits)	-		•	•	247
15	SBM	BIN multiplication (24 bits)	-		•	•	249
16	SBW	BIN division (24 bits)	-		•	•	251
17	SCA	Write to internal clock element	-		•	•	253
18	SCB	Read from internal clock element	-		•	•	254
19	SMOVK	Data transfer from memory to memory	-		•		255
20	BBO	Display of bar	-	•			257
21	BCR	Display of circle	-	•			259
22	BWK	Display of frame	-	•			261
23	SDE	Erasure of file	-	•			263
24	SLD	Read of file	-	•			264
25	SLE	Lighting of operation panel LED	-	•			266
26	SOK	Check of operation panel input	-	•			267
27	SST	Storage of file	-	•			268
28	SPC	Discrimination of programmable controller CPU	-			•	270
29	SRK	Read of sequence program in ladder mode	-			•	271
30	SWK	Write of sequence program in ladder mode	-			•	273
31	SRI	Read of sequence program in list mode	-			•	275
32	SWI	Write of sequence program in list mode	-			•	277
33	SKC	Programmable controller run/stop check	-			•	279

Table 2 List of System Subroutines Which Are Usable for BASIC Command

	System Subroutine	Function	Type	Applicable Model			Reference Page
				GPC1	GPC11	KD51	
34	SKR	Remote run of programmable controller CPU	-			•	280
35	SKP	Remote stop of programmable controller CPU	-			•	281
36	SKI	Interval setting of access time to programmable controller CPU	-			•	282
37	SOPEN	Opening of channel of K30RSF	-	•	•		283
38	SRB	Receiving of specified byte length of data send to specified channel	A	•			285
			B		•	•	287
39	SWB	Sending of specified byte length of data from specified channel	A	•			289
			B		•	•	291
40	SRC	Read of the number of bytes of data received by specified channel	-		•	•	294
41	SRF	Read of the number of vacant bytes in receive buffer of specified channel	-		•	•	295
42	SHX	Control of send/receive data of specified channel by Xon/Xoff codes	-		•	•	296
43	SHD	Control of send/receive data of specified channel by DR terminal	-		•	•	297
44	SAE	Conversion of all send/receive data of specified channel to EBCDIC code	-		•	•	298
45	SEA	No code conversion of send/receive data of specified channel	-		•	•	299
46	SEN	END instruction search	-			•	300
47	SBU	Read, write and verify of data between buffer memory of K30MU10 and bubble memory	-		•		301
48	SBE	Read of error data of K30MU10	-		•		302
49	SITX	Read of interruption input data of K30H41	-		•		303
50	SITY	Clear of interruption input data of K30H41	-		•		304
51	SITA	Setting of subroutine for interruption of K30H41	-		•		305
52	SITB	Clear of setting of subroutine for interruption of K30H41	-		•		307

Note: System subroutines in Table 2 can be used for BASIC command and microcomputer program.

Table 2 List of System Subroutines Which Are Usable for BASIC Command (Continued)

	System Subroutine	Function	Type	Applicable Model			Reference Page
				GPCI	GPCII	KD51	
1	SCALL	Call of subroutine in specified address of specified channel	A	•			309
			B		•		310
2	SALU	Operation of 32-bit floating-point type data	—	•	•		311
3	SMOV	Transfer of data from memory to memory	—	•	•		313
4	SRD1	Read of one-byte data from specified channel	—	•	•		314
5	SRD2	Read of two-byte data from specified channel	—	•	•		315
6	SWR1	Write of one-byte data to specified channel	—	•	•		316
7	SWR2	Write of two-byte data to specified channel	—	•	•		317
8	STIME	Interruption of execution for specified period of time	—	•	•		318
9	SQUIT	End of program run	—	•	•		319
10	SRSFCO	Direct write of command to K30RSF	—		•		320
11	SRSFST	Read of K30RSF status	—		•		321
12	SRSFWR	Direct write of data to K30RSF	—		•		323
13	SRSFRD	Direct read of data from K30RSF	—		•		322

Note: System subroutines in Table 3 can be used only for microcomputer program.

Table 3 List of System Subroutines Which Are Not Usable for BASIC Command

CONTENTS

1. FEATURES OF GPC-BASIC

1 ~ 13

1.1 Software configuration	3
1.2 Communication with KCPU	4
1.3 Multi task	6
1.4 Link with host computer	13

2. BASIC OF GPC-BASIC

15 ~ 53

2.1 General description of GPC-BASIC	16
2.1.1 Operation mode	16
2.1.2 Form of line	17
2.1.3 Applications of keys	18
2.2 Set of characters	19
2.3 Constants	19
2.3.1 Character string constant	19
2.3.2 Numeric constant	20
2.4 Variables	22
2.4.1 Basic variable	22
2.4.2 "@" variable	23
2.4.3 Indirect variable	24
2.4.4 Character string variable	27
2.5 Expressions and operations	29
2.5.1 Arithmetic operation	30
2.5.2 Comparing operation	32
2.5.3 Logical operation	33
2.5.4 Built-in functions	35
2.5.5 Character string operation	36
2.5.6 Bit operation	37
2.5.7 Form of assignment statement	42
2.5.8 Comparison of expressions	45
2.6 How to prepare program	48
2.6.1 How to prepare program	48
2.6.2 Correction of program	49

3. GPC-BASIC COMMANDS **55 ~ 213**

- 3.1 Key board commands. 56
- 3.2 Program commands 76
 - 3.2.1 Program commands which are not functions 77
 - 3.2.2 Program commands which are functions 126
- 3.3 Z commands. 134
 - 3.3.1 Z commands which are not functions 135
 - 3.3.2 Z commands which are functions 152
- 3.4 Built-in functions 167
- 3.5 Graphic commands 182

4. SYSTEM SUBROUTINES **215 ~ 323**

- 4.1 What is system subroutine? 216
- 4.2 Usage of system subroutine 217
- 4.3 Subroutines which can be used for BASIC and microcomputer programs 222
- 4.4 Subroutines which can be used only for microcomputer program 308

5. APPENDIX **325 ~ 342**

- 5.1 Error messages displayed during programming. 326
- 5.2 Cautions during preparation of BASIC program. 327
- 5.3 Program prepared with microcomputer instruction 333
- 5.4 Addresses of system subroutines. 340

A decorative rectangular border with a repeating floral or scrollwork pattern surrounds the title text.

FEATURES OF GPC-BASIC

1. FEATURES OF GPC-BASIC

GPC-BASIC is a language dedicated to the MELSEC-K series and based on the BASIC language, to which commands have been added to allow communication with a programmable controller. Therefore, GPC-BASIC has the following features which the BASIC language does not have.

1. Communication can be made with a programmable controller by Z commands.
2. A maximum of eight tasks (programs) can be prepared.
3. A plurality of prepared tasks can be run by multi tasking.
4. Allows link with a host computer.

1.1 Software Configuration

Fig. 1.1 shows the software configuration of tasks (user programs) which have been prepared with GPC-BASIC or microcomputer program.

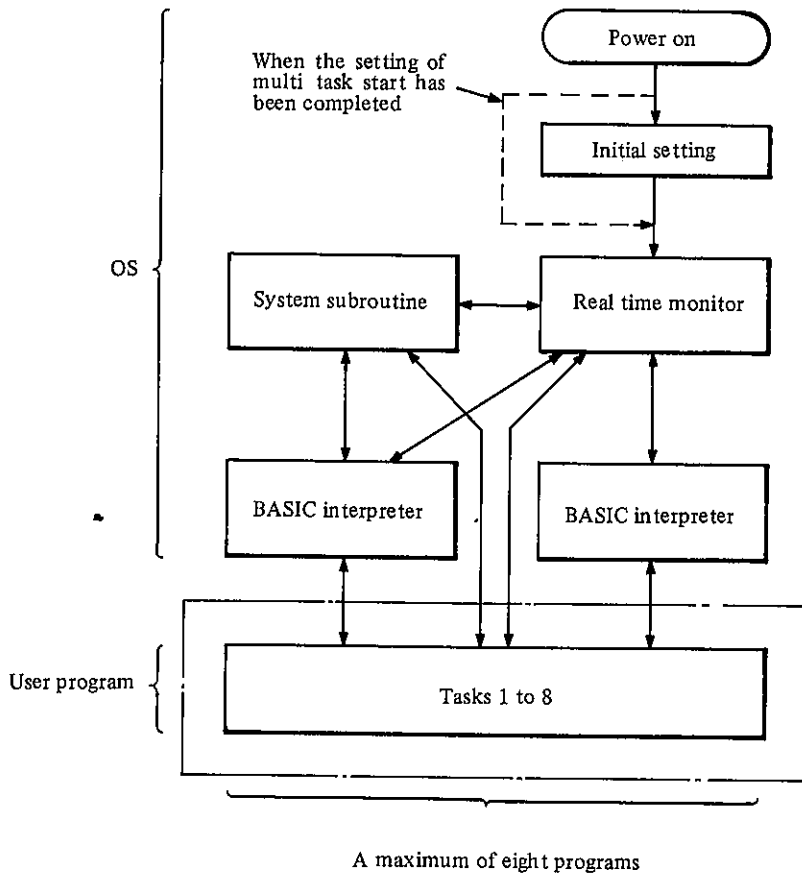


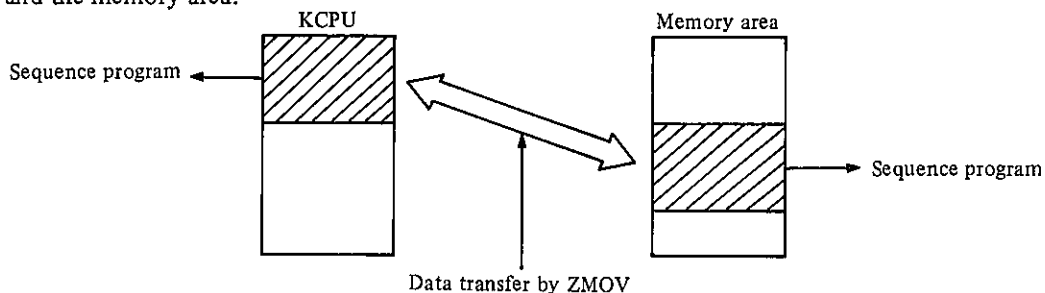
Fig. 1.1 Software Configuration

1.2 Communication with KCPU

The BASIC program permits communication with the KCPU as described below:

(1) Transfer of sequence program

The BASIC program allows the transfer of data between the sequence program area of KCPU and the memory area.

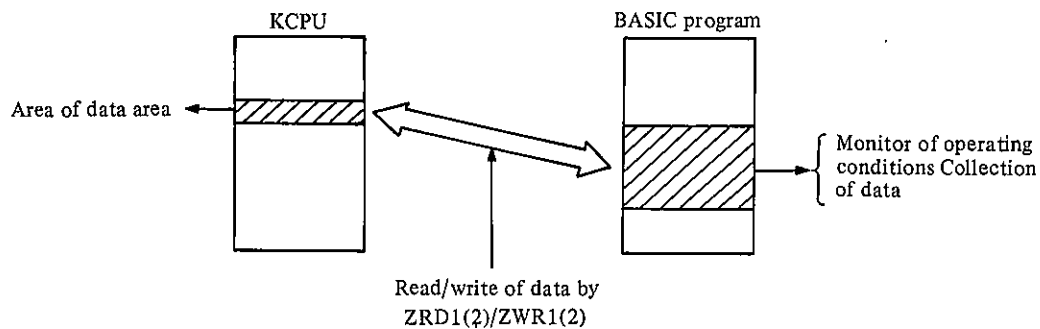


(2) Read/write of data in data area of KCPU

Read/write of data of KCPU can be performed with the following devices.

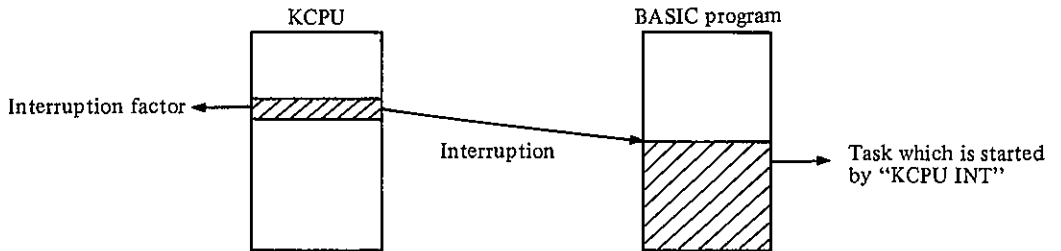
Read..... { ON/OFF data of X, Y, M, T, C, F and K
 Temporary value of T and C
 Content of D

Write..... { ON/OFF data of Y, M, T, C, D, F and K
 Temporary value of T and C
 Content of D



(3) Interruption by KCPU

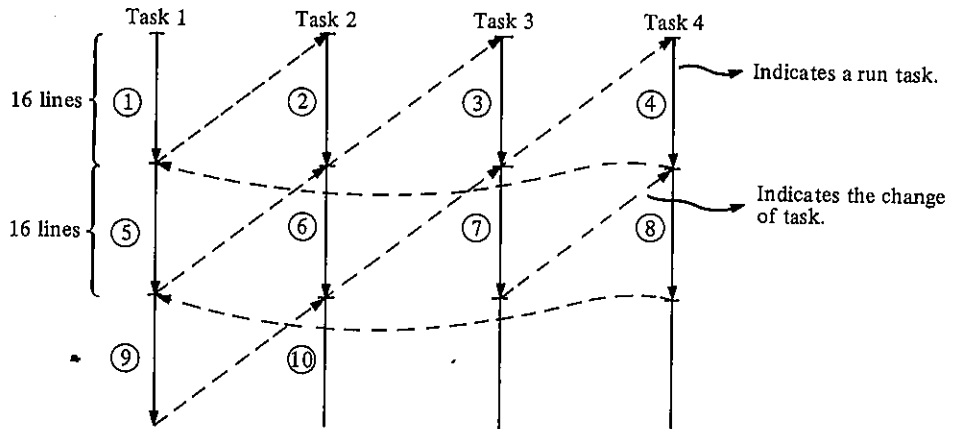
With the start condition of task set to "KCPU INT" during the setting of multi task, the task can be started when the interruption factor by KCPU holds. ("KCPU INT" can be set only for one task.)



1.3 Multi Task

- (1) A maximum of eight tasks (program numbers 1 to 8) can be prepared by the equipment which uses GPC-BASIC. The plural tasks prepared can be run by multitasking with real time monitor.
- (2) Basically, multi task is run by changing the tasks in order of task 1 to 8 each time 16 lines are executed. However, when the task changing factor has occurred during run of task, the task is changed and another task is run if 16 lines are not executed.

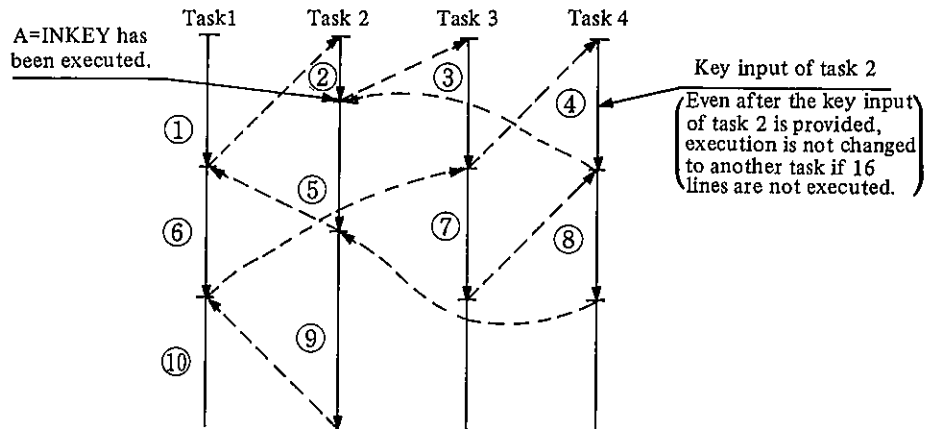
1) Basic run of task



- Tasks 1 to 4 are run by 16 lines in order of ① to ⑩.

2) Task changing factor has occurred

- Task changing factor { BASIC command in Table 1.1 has been executed
System subroutine in Table 1.2 has been executed



- Tasks 1 to 4 are run in the following order:

- ① 16 lines are executed.
- ② Executed up to A=INKEY (task changing factor)
- ③ 16 lines are executed.
- ④ 16 lines are executed. (Assume that data of task 2 has been entered.)
- ⑤ Lines following A=INKEY ("16 lines" minus "the number of lines executed in ②") are executed.

- ⑥
- ⑦
- ⑧
- ⑨
- ⑩
- ⋮

Execution is started in order of tasks which have completed the execution of 16 lines.



Run of task stops in order of lower to higher numbers.

	BASIC Command			Wait Factor	Remarks
1	INPUT INKEY			Wait for key input interruption.	When there is key input in the buffer, execution is continued without changing task.
2	ZTIME			Wait for 10ms internal timer interruption.	
3	END			Wait until start condition holds. { Wait for KCPU interruption. { Wait for 10ms internal timer interruption.	
4	CLS	COLOR	LCOPY	Wait for 10ms internal timer interruption	<ul style="list-style-type: none"> • Only LPRINT and LCOPY for GPC1, and wait factor is wait for sending/receiving completion interruption. • Applicable to KD51 are only those usable for BASIC command.
	LOCATE	LPRINT	PRINT		
	ZDSP	ZCOFF	ZCON		
	ZCRV	ZNOR	LINE		
	CIRCLE	PSET	PRESET		
	LTYPE	GCOLOR	FTYPE		
	MODE	FSTYLE	ORIGIN		
	WINDOW	VIEW			
	OPEN CLOSE	(Only GPC11)			

Table 1.1 BASIC Commands Which Change Task

CAUTION

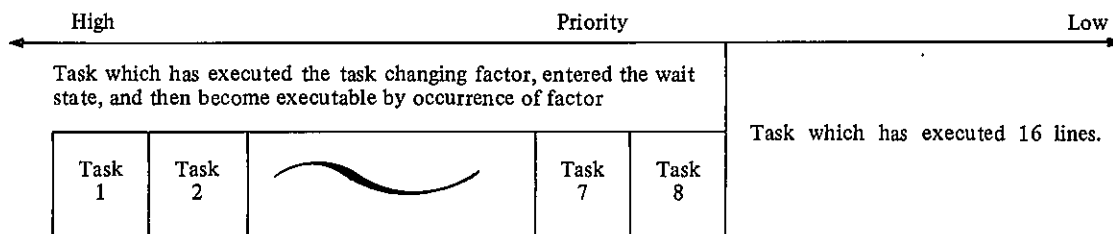
When the buffer of K31PST is not full in the KGPC11, the task is not changed and execution is continued.

		Wait Factor	Remarks
1	STIME (Suspension)	Wait for 10ms internal timer interruption	Same as ZTIME command - of BASIC
2	SMOVK (Data transfer from memory to memory)		Only for K3NCPU
3	SRB (Block data receiving) SWB (Block data sending)	Wait for 10ms internal timer interruption.	In GPC1, wait for sending/receiving completion interruption.
4	SBU (Bubble)	Wait for K30MU10 interruption. Wait for 10ms internal timer interruption.	
5	STIX (Interruption servicing)	Wait for K30H41 interruption.	If interruption has occurred before the execution of STIX, execution is continued without changing task.
6	SQUIT (Run completion)	Wait until start condition holds. { Wait for KCPU interruption. { Wait for 10ms internal timer interruption.	

Table 1.2 System Subroutines Which Change Task

(3) The priority of task is as follows:

- 1) { Tasks which have executed the task changing factor, have entered the wait state, and then have become executable by occurrence of factor } ⇒ { The lower the task number is, the higher the priority is. }
- 2) { Tasks which have executed 16 lines of BASIC program. } ⇒ { The task, which has entered the wait state earlier after execution of 16 lines, has higher priority. }
- 3) { Tasks described in ① and ②. } ⇒ { Tasks described in ① has higher priority than tasks described in ②. }



(4) When the currently run task is placed in wait state (when 16 line has been executed or the command in Table 1.1 or Table 1.2 has been executed), the task is changed as described below:

- 1) $\left[\begin{array}{l} \text{Therefore exist executable tasks.} \end{array} \right] \Rightarrow \left[\begin{array}{l} \text{Tasks are run in order of higher priority desc-} \\ \text{cribed in (3).} \end{array} \right]$
- 2) (There does not exist an executable task.) \Rightarrow (OS waits until one of tasks become executable.)

CAUTION

Task is not changed when a task with high priority is placed in executable state, but is changed only when the currently run task is placed in wait state.

(5) There are the following three start conditions of task. (Set the start condition of task during the setting of multi task.)

- 1) Power on: When run of multi task is started, the start condition of task holds. The task of which start condition has held is run in order of priority.
- 2) KCPU interruption: Start condition holds when interruption input is given by KCPU.
- 3) Real time interruption: ... Start condition holds at set intervals of time.

CAUTION

- (1) KCPU interruption can be set for only one task.
- (2) Set real time interruption so that real time interruption is not caused during run of task which has been started by real time interruption.

EXAMPLE

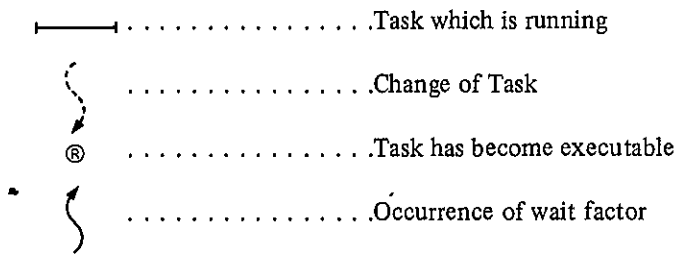
Progress of multi task run

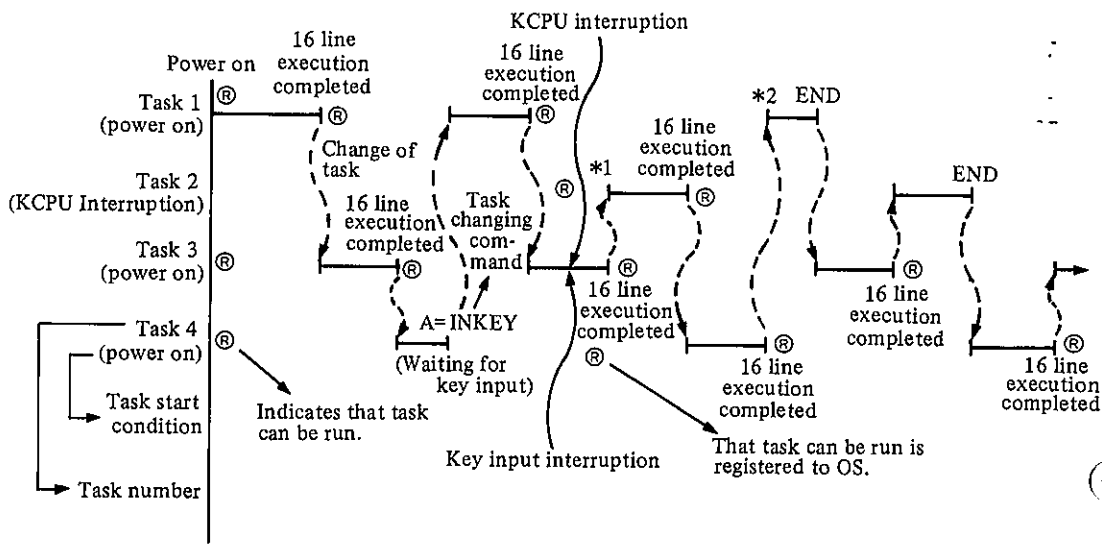
Fig. 1.2 shows a progress example of multi task run with four tasks.

- Assume that the start condition of each task is as follows:

Task 1 Power on
Task 2 KCPU interruption
Task 3 Power on
Task 4 Power on

- Assume that symbols in Fig. 1.2 are as follows:





Low ← Priority → High

Task which is being run	⊗	1	3	4	1	3	2	4	1	3	2	4
Task which can be run by start condition or occurrence of factor	1 3 4	3	4			4 2 4	4					
Task which can be run by completion of 16 line execution			1	1 3	3	1 1 1	1 3	1 3 2	3 2 4	2 4	4 3	3

Tasks which can be run during power on

*1: Since task 2 has higher priority over task 4, task 2 is run.
 *2: Among tasks which can be run by completion of 16 line execution, the task which has become executable earliest is run.

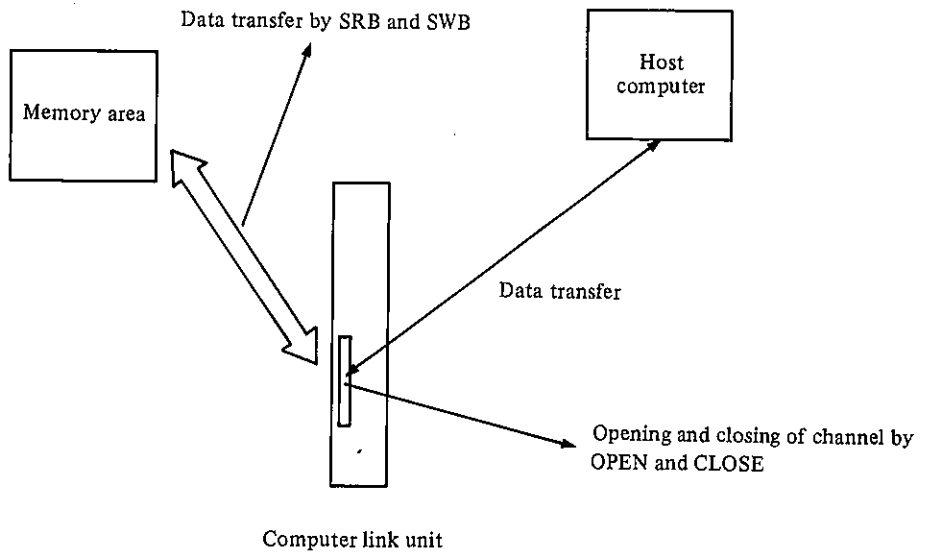
Fig. 1.2 Task Changing Diagram (for GASIC program)

1.4 Link with Host Computer

By use of OPEN and CLOSE of BASIC commands, data communication can be made with a host computer by the BASIC program.

OPEN: Opens the specified channel and permits sending and receiving operations.

CLOSE: Closes the specified channel.



CAUTION

1. For the GPC1 or GPC11, load the RS-232C interface unit (K30RSF) to connect it with the host computer.
2. The KD51 incorporates the RS-232C interface and RS-422 interface.



A decorative rectangular border with a repeating floral or scrollwork pattern surrounds the title text.

BASIC OF GPC-BASIC

2. BASIC OF GPC-BASIC

2.1 General Description of GPC-BASIC

This section explains the general description of GPC-BASIC. Since some parts of general description differ depending on models, refer to the Instruction Manual for details.

2.1.1 Operation mode

When BASIC is started, the screen displays $\text{>}\blacksquare$. This $\text{>}\blacksquare$ indicates a state in which a command can be accepted. (This state is referred to as a "command accept state".) Namely, the user can enter all BASIC commands. There are the following two methods for running a program.

- (1) To run a program simultaneously with data entry through keys

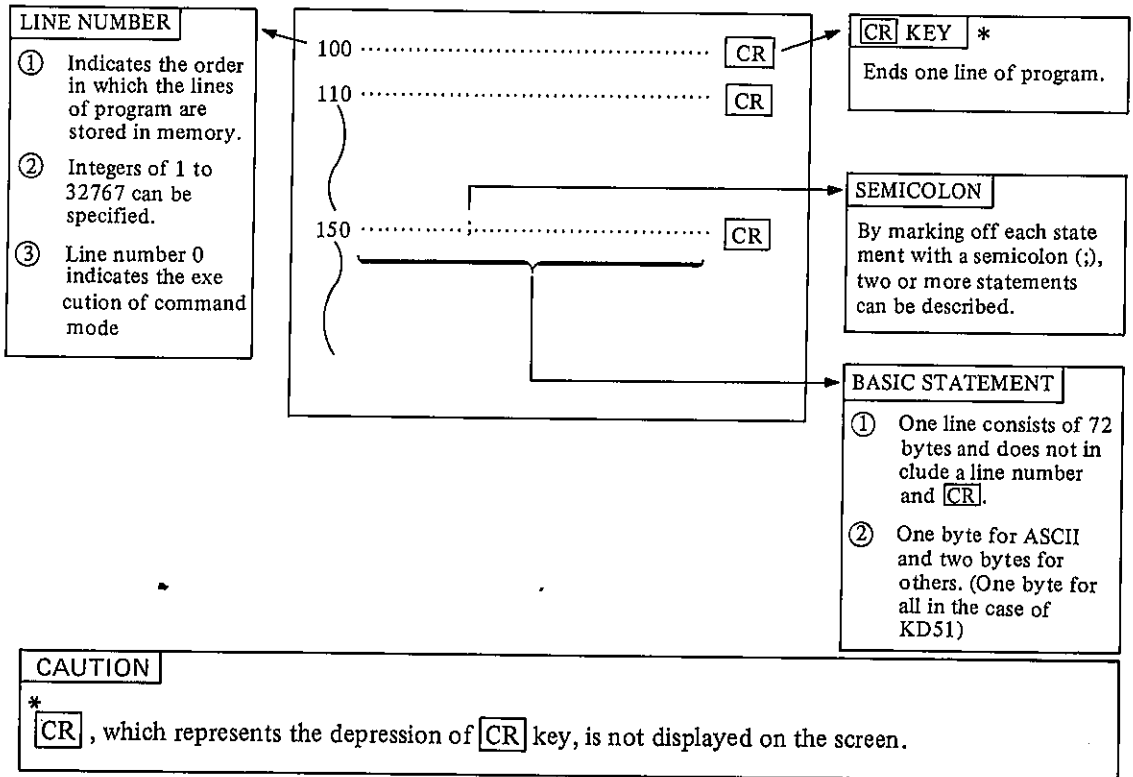
When a program is entered without providing line numbers for the statements and commands of BASIC, BASIC immediately executes the entered statement and command. In this case, the entered program does not remain in the memory. This is referred to as the "execution of command mode".

- (2) To run a program after completion of data entry through keys

When a program is entered with the statements and commands of BASIC provided with line numbers, BASIC does not execute the entered statement and command immediately, and stores them together with line numbers in the memory. The program stored in the memory is run by RUN command. This is referred to as the "execution of program mode".

2.1.2 Form of line

The program of BASIC consists of lines. The form of line is as shown below:



2.1.3 Applications of keys

The applications of keys, which are often used for programming, are shown in the following table:

Key Name	Application	Key Code
CR	<ul style="list-style-type: none"> ○ During programming, the operation mode returns to the command accept state. ● The entry of one line has been completed. 	0DH
RETURN	<ul style="list-style-type: none"> ● Correction has been completed by the EDIT command. ○ When the AUTO command is used, the next line number is generated. 	
CONT / C *	<ul style="list-style-type: none"> ○ Used when it is desired to suspend the run of program halfway. ○ When the keys are pressed before completion of entry of one line, the line is not stored in the memory. 	03H
CTRL / C	<ul style="list-style-type: none"> ○ When the AUTO command is used, the same line number is generated. 	
ESC	<ul style="list-style-type: none"> ○ Used to stop the entry of one line during entry of the line. ○ *Stops the execution of AUTO command. 	1BH
CAN	<ul style="list-style-type: none"> ○ Used to erase one line of program during entry of the line. 	18H
DEL	<ul style="list-style-type: none"> ○ A character, which is located before the cursor, is erased. 	7FH
SPACE	<ul style="list-style-type: none"> ○ A space, which is as large as the cursor, is provided. 	20H

*: **CONT** / **C** indicates that the **CONT** and **C** keys are pressed at the same time.

NOTE

The key name may differ from the one indicated in the above table. In such a case, judge the application from the key code in the above table.

2.2 Set of Characters

The set of characters, which can be used for GPC-BASIC, are as follows:

- ASCII characters $\left\{ \begin{array}{l} \text{Numerals (10 characters from 0 to 9)} \\ \text{Letters (26 capital letters from A to Z)} \\ \text{Symbols} \end{array} \right.$
- Special characters Letters and symbols

2.3 Constants

The constant is an actual value used to execute GPC-BASIC. The types of constants are as follows:

Constant	{	Character string constant	Character string enclosed by " "	
		Numeric constant	Integer constant	Integers of -32767 to 32767
			Fix-point constant	Real numbers of 2.7×10^{-20} to 9.2×10^{18}
			Floating-point constant	Number, expressed in exponent form, of 2.7×10^{-20} to 9.2×10^{18}
		Hexadecimal constant	Hexadecimal number preceded by \$	

2.3.1 Character string constant

The character string constant is a string of arbitrary characters enclosed by " ".

EXAMPLE

```
"HELLO"
"$250000.00"
"OPERATION START"
```

CAUTION

1. Be sure to use the double quotes ("....") in pairs.
2. When (") is provided in an odd number, the program does not function normally.

EXAMPLE

Duplicated ➔ "...." "

Missing ➔ "....

2.3.2 Numeric constant

The numeric constant is a positive or negative number. This constant cannot include a comma. There are the following four types of numerical constants. (When the constant is positive, the “+” symbol can be omitted.)

(1) Integer constant

- An integer ranging from -32767 to $+32767$.

EXAMPLE

A=100

A=+123. Symbols of A=123 and “+” can be omitted.

A=-32767

(2) Fixed-point constant (Significant digits: 7 digits)

- A positive or negative real number (number including a decimal point). This constant also includes an integer except the integer constant in (1).
- The fixed-point constant can represent a value from 2.7×10^{-20} to 9.2×10^{18} .

EXAMPLE

A[0]=100.0

A[0]=-123.21

(3) Floating-point constant (Significant digits: 7 digits)

- A positive or negative number represented in exponent form.
- This constant comprises an integer or a fixed-point number (fixed point part) which is followed by a letter E and then by an integer (exponent part).
- The floating-point constant can represent a value from 2.7×10^{-20} to 9.2×10^{18} .

EXAMPLE

0.2359887E - 04 \Rightarrow 0.00002359887

-0.2359000E 10 \Rightarrow -2359000000

(4) Hexadecimal constant

- Hexadecimal numbers (0 to 9, A to F) provided with \$ at the beginning, which can represent four digits from 0000 to FFFFH.

EXAMPLE

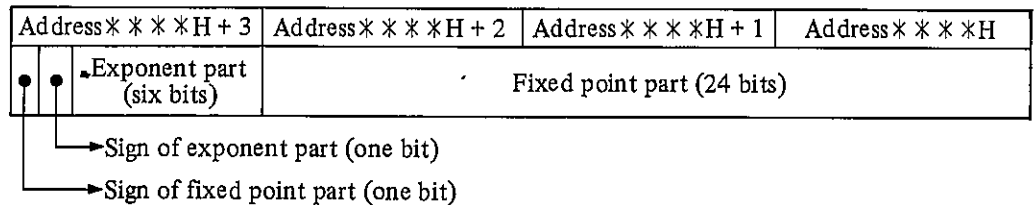
\$76
\$32F
SA6C5

REFERENCE

Floating point number

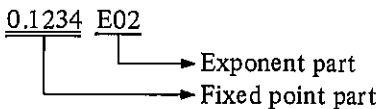
This is a system of notation which represents a real number by an exponent part and a fixed point part. To obtain the numeric value of floating point number, multiply a numeric value, which is obtained by raising a certain cardinal to the power indicated by the number of an exponent, by the number of a fixed point part.

In GPC-BASIC, the floating point consists of 32 bits (four bytes) as shown below:

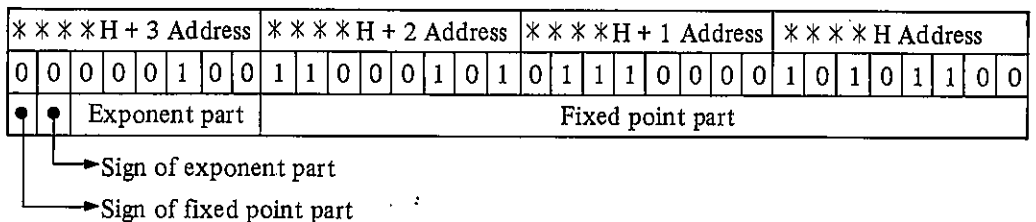


EXAMPLE

A number which is normally written "12.34" is represented in floating point as shown below:

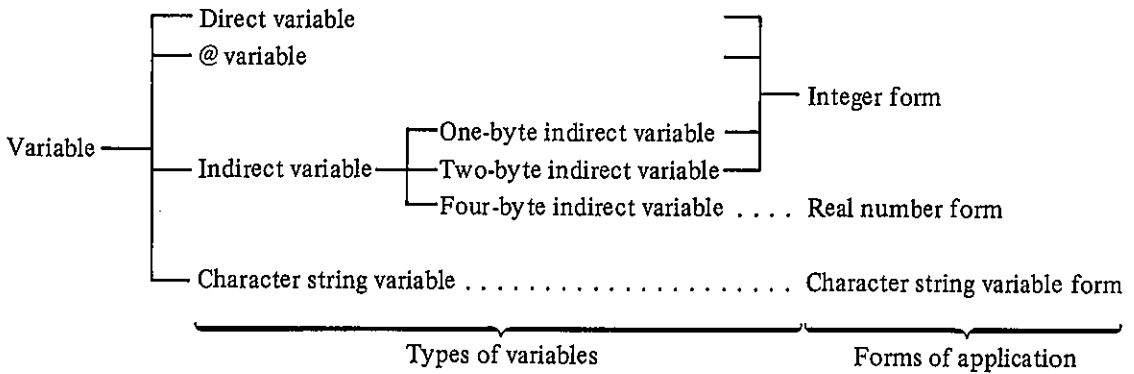


This number is stored in 32 bits (four bytes) as shown below:



2.4 Variables

- (1) Variables are used for the storage or operation of values which are used in the program of GPC-BASIC.
- (2) As variables, letters of A to Z are used.
- (3) Numeric values usable as variables are -32767 to 32767 .
- (4) The memory addresses used for variables are in the same channel as those of the BASIC program.



2.4.1 Direct variable

In the case of direct variable, a numeric value is directly assigned to the variable.

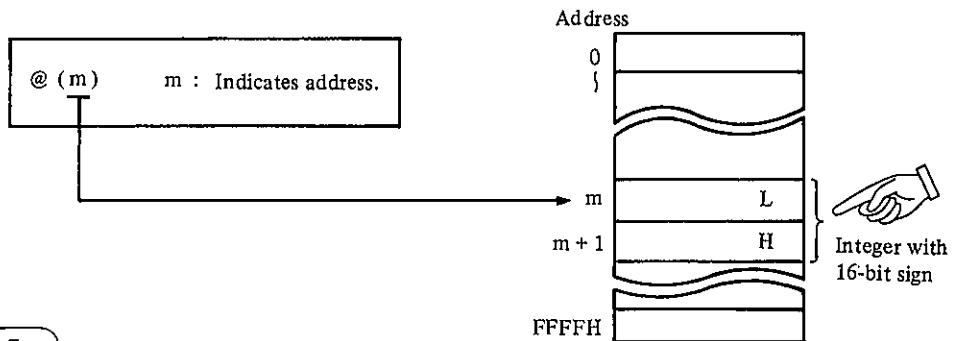
Variable = integer

EXAMPLE

- (1) $A = \$E000$ Variable A is set to E000H.
- (2) $B = 2$ 2 is assigned to variable B.

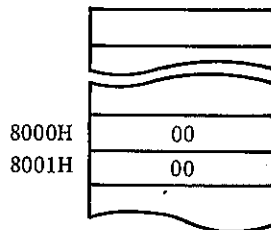
2.4.2 "@" variable

Two bytes from the addresses (words) in the specified memory space are handled as an integer with 16-bit sign.

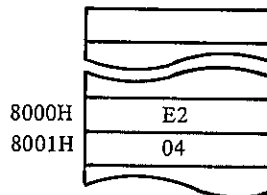


EXAMPLE

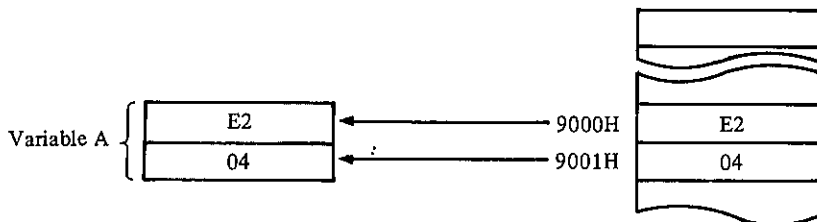
(1) $@\$(8000) = 0$ 0 is stored in memory addresses 8000H and 8001H.



(2) $@\$(8000) = 1250$ 1250 (04E2 in hexadecimal number) is stored in memory addresses 8000H and 8001H.



(3) $A = @\$(9000)$ Values in memory addresses 9000H and 9001H are assigned to variable A. [Assume that 1250 (04E2 in hexadecimal number) is stored in variable A.]



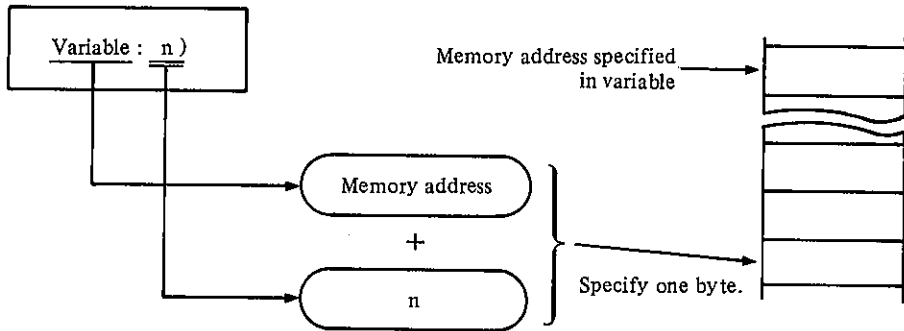
2.4.3 Indirect variable

The indirect variable handles the memory of one byte, two bytes (one word) or four bytes (only real number) which are indicated by offset values (-32767 to 32767) which use the direct variables as bases.

Indirect variables {

- One-byte indirect variable. Handles integers of 0 to 255.
- Two-byte indirect variable. Handles integers of -32767 to 32767 .
- Four-byte indirect variable Handles real numbers of 2.7×10^{-20} to 9.2×10^{18} .

(1) One-byte indirect variable



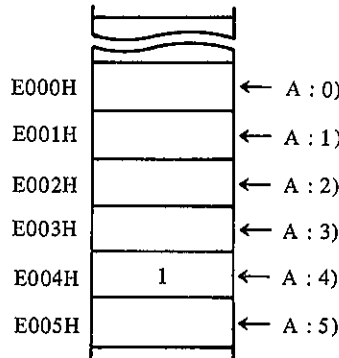
- 1) Specify one byte from the memory addresses of (value of variable) + (n).
- 2) Numeric values useable as variables are integers of 0 to 255.

EXAMPLE 1

To write specified numeric value to specified address

```

100  A = $E000
110  B = 2
120  A : B + 2) = 1
130  END
    
```

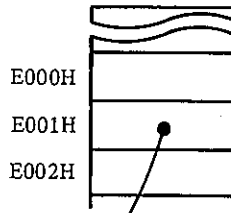


EXAMPLE 2

To read the content of specified address

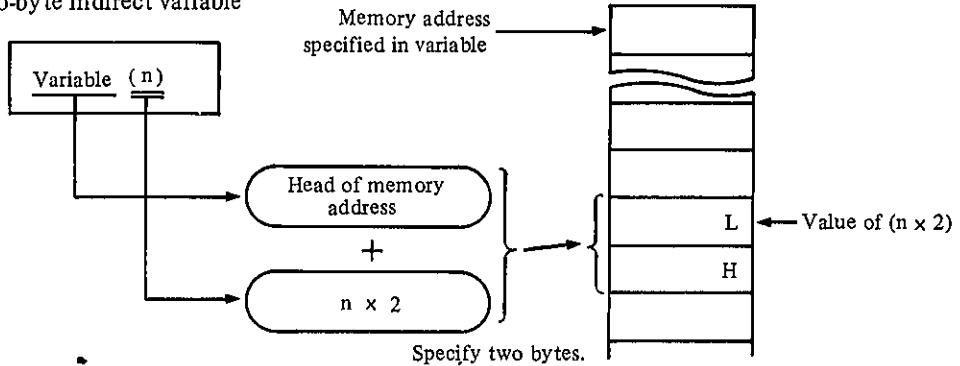
```

100  A = $E000
110  PRINT A : 1)
120  END
    
```



Memory contents are displayed on the screen.

(2) Two-byte indirect variable



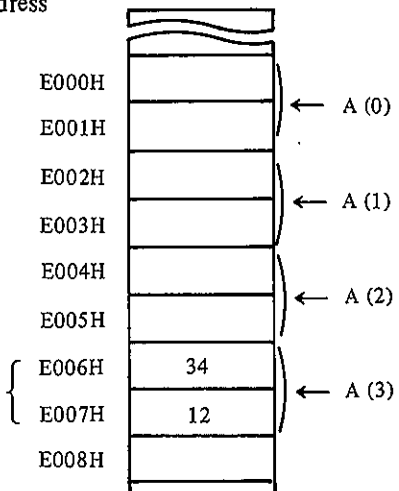
- 1) Specify two bytes from the memory addresses of (value of variable) + (n x 2).
- 2) Numeric values useable as variables are integers of -32767 to 32767.

EXAMPLE 1

To write specified numeric value to specified address

```

100  A = $E000
110  B = 2
120  A(B + 1) = $1234
130  END
    
```

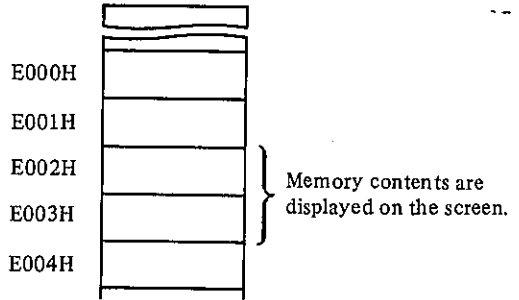


EXAMPLE 2

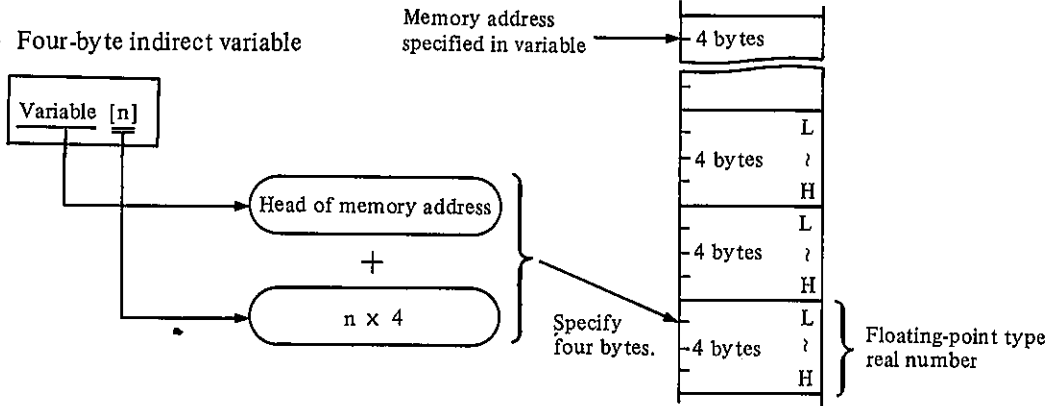
To read the contents of specified addresses

```

100  A = $E000
110  PRINT A(1)
120  END
    
```



(3) Four-byte indirect variable



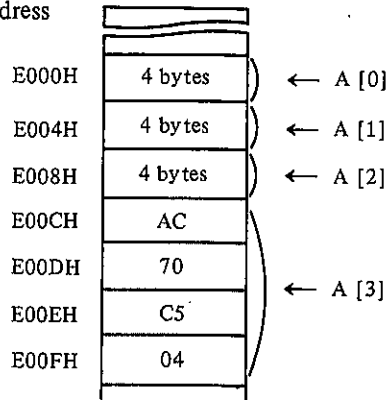
- 1) Specify four bytes from the memory addresses of (value of variable) + (n x 4).
- 2) Numeric values usable as variables are real numbers of 2.7×10^{-20} to 9.2×10^{18} .
- 3) The contents of memory are of floating-point type.

EXAMPLE 1

To write specified numeric value to specified address

```

100  A = $E000
110  B = 2
120  A[B + 1] = 12.34
130  END
    
```



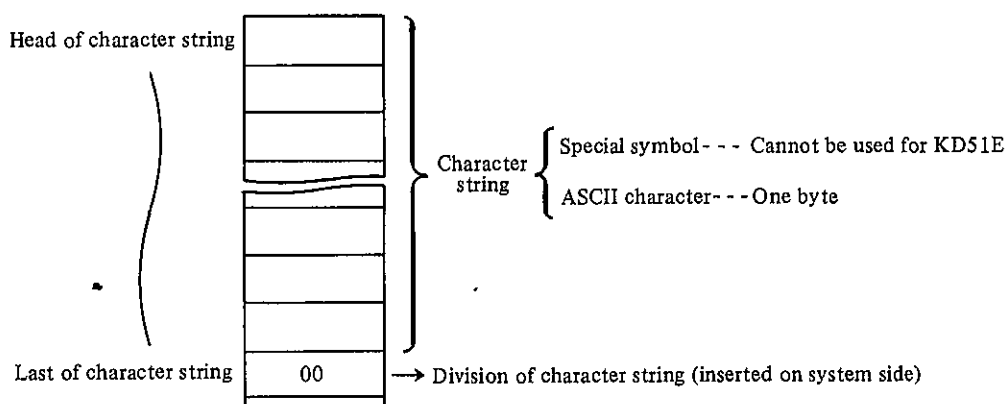
2.4.4 Character string variable

- 1) By enclosing a character string with " ", the character string can be assigned to a variable.
- 2) The format of character string variable is as follows:

A\$ For direct variable
 A\$(2) For two-byte indirect variable

- 3) Structure of character string

The structure of character string is as shown below:

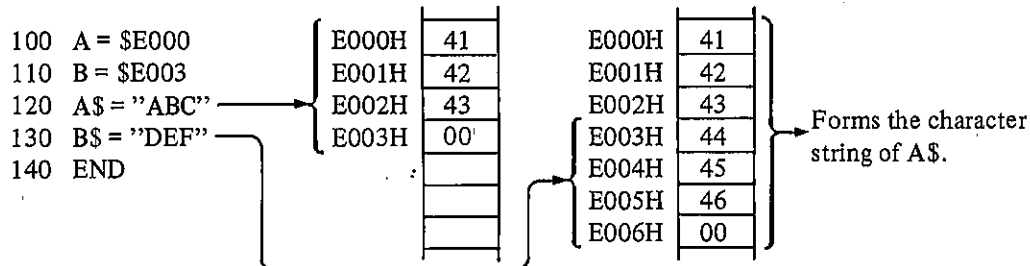


CAUTION

At the end of character string, the division code of character string "00" is inserted on the system side. Therefore, caution should be exercised when character strings are stored consecutively.

EXAMPLE

The following shows the case where character strings have been stored consecutively without giving consideration to the division of character strings.



EXAMPLE

```

100  A = $E000 .....Variable A is set to address E000H.
110  B = $6000
120  A $ = "ABCDE" .....ABCDE is written in key codes to E000H and follow-
130  B(2) = $F000 .....ing addresses. Note 1
140  B$(2) = "MITSUBISHI ELECTRIC" .....MITSUBISHI ELECTRIC is written in key codes to
                                         F000H and following addresses.
150  PRINT $A .....Contents of variable A are displayed in hexadecimal.
160  PRINT AS .....Contents of E000H and following addresses are displa-
170  PRINT $B(2) .....yed in characters corresponding to key codes.
180  PRINT B$(2)
190  END
OK
>RUN
E000 .....Display by instruction of line number 150.
ABCDE .....Display by instruction of line number 160.
F000 .....Display by instruction of line number 170.
MITSUBISHI .....Display by instruction of line number 180.

```

The contents of memory beginning with address E000H are as follows:

E000H	41	← Key code of "A"
E001H	42	← Key code of "B"
E002H	43	← Key code of "C"
E003H	44	← Key code of "D"
E004H	45	← Key code of "E"
E005H	00	← Division of character string

2.5 Expressions and Operations

- (1) An expression is a combination of constants and variables, which are connected by operators to obtain a character string constant, numeric constant, variable or certain value.

EXAMPLE

"MITSUBISHI ELECTRIC"Character string constant
5.64Numeric constant
A=\$E000.....Variable
10 + 10/3 }
A + B/C }Connection by operators
TAN(30).....Function

(2) Types of operation

- 1) Arithmetic operation
- 2) Comparing operation
- 3) Logical operation
- 4) Function
- 5) Character string operation
- 6) Bit operation
- 7) Assignment statement
- 8) Comparison

(3) Priority of operation

High	(1) Portion enclosed by parentheses
↓	(2) Variable, function
	(3) Multiplication (*), division (/), bit operation, logical product AND (&), remainder (%), exponent (^)
	(4) Addition (+), subtraction (-), logical sum OR (!), exclusive logical sum EXOR (^)
	(5) Negative sign (-), negation (#)
Low	(6) Comparison (<, >, =, <=, >=, #)

2.5.1 Arithmetic operation

Arithmetic operators are as follows:

<u>Operator</u>	<u>Meaning</u>	<u>Example</u>
$*$, $/$	Multiplication, division	$* Y$, X/Y
\wedge	Exponent	$X \wedge Y$
$+$, $-$	Addition, subtraction	$X + Y$, $X - Y$
$-$	Negative sign	$-Y$

To change the order of operation, use parentheses. When part of the expression is in parentheses, operation is performed beginning with the inside of the parentheses. When operators are provided consecutively, each operator should be divided by parentheses.

Examples of mathematical expressions in algebra and corresponding representations by BASIC are shown below:

EXAMPLE

<u>Representation in algebra</u>	<u>Representation by BASIC</u>
$X + 2Y$	$X + 2 * Y$
$X - \frac{Y}{Z}$	$X - Y / Z$
$\frac{XY}{Z}$	$X * Y / Z$
$\frac{X + Y}{Z}$	$(X + Y) / Z$
$(X^2)^Y$	$(X \wedge 2) \wedge Y$
X^{Y^2}	$X \wedge (Y \wedge Z)$
$X(-Y)$	$X * (-Y)$

CAUTION

1. Since the result of exponent calculation is obtained as a real number, the representation cannot be $Z = X \wedge Y$. In this case, therefore, use the form of four-byte indirect variable, for instance:

$$A = \$E000$$
$$A[0] = X \wedge Y$$

The four-byte indirect variable form may also be used for variables corresponding to X and Y.

2. The result of negative exponent operation is always "0".

$$A[0] = (-2)^2 \Rightarrow 0$$

(2) Remainder operation

The remainder operation is represented by an operator %, and the remainder of integer operation is given as an integer.

EXAMPLE

$$A = 10\%3 \quad \Rightarrow \quad 1 \dots \dots \dots 10 \div 3 = 3 \text{ Remainder } \underline{1}$$
$$A = 785\%100 \quad \Rightarrow \quad 85 \dots \dots \dots 785 \div 100 = 7 \text{ Remainder } \underline{\underline{85}}$$
$$A = (-10)\%3 \quad \Rightarrow \quad -1 \dots \dots \dots -10 \div 3 = -3 \text{ Remainder } \underline{\underline{-1}}$$

CAUTION

In the remainder operation of expression of which left member is of integer type as shown below, calculation is performed regarding the decimal point as a bit operator.

$A=78.5\%10 \Rightarrow$ Remainder obtained by dividing the fifth bit of address
78 by 10
↓
Regarded as bit operator

(3) Division in the case of overflow or when divisor is 0 (only for real number calculation)

When a divisor is 0 in a division during execution of an expression, 0 is given as the division result and the program continues.

When overflow occurs, 0 is given as the operation result and the program continues.

2.5.2 Comparing operation

- (1) The comparing operator is used to compare two values. The result of comparison is represented by "true" (1) or "false" (0), and used to change the flow of program by use of condition judging statement, etc. (See IF command in page 91)

<u>Operator</u>	<u>Meaning</u>	<u>Example</u>
=	IS EQUAL TO	X = Y
#	IS NOT EQUAL TO	X # Y
<	IS LESS THAN	X < Y
>	IS GREATER THAN	X > Y
<=	IS NOT GREATER THAN	X <= Y
>=	IS NOT LESS THAN	X >= Y

(2) The equal sign “=” is also used to assign a value to a variable.

2.5.3 Logical operation

(1) The logical operators are as follows:

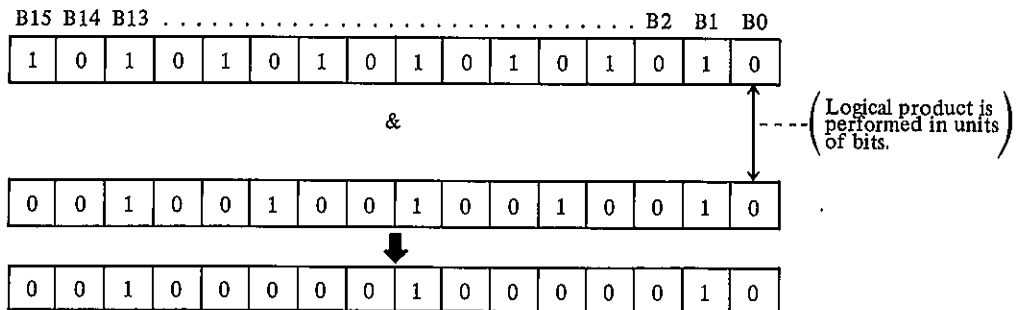
Operator	#		&			!			≠ (\)		
Meaning	Negation		Logical product			Logical sum			Exclusive logical sum		
Example	X	#X	X	Y	X & Y	X	Y	X ! Y	X	Y	X ≠ Y
	0	1	0	0	0	0	0	0	0	0	0
	1	0	0	1	0	0	1	1	0	1	1
			1	0	0	1	0	1	1	0	1
			1	1	1	1	1	1	1	1	0

(2) The logical operation performs the bit operation or Boolean algebra operation of integers.

(3) The logical operation is performed in units of corresponding bits after converting an operand into an integer (represented by 2's complement for negative number) in the range of -32767 and +32767.

EXAMPLE

The following shows an example of logical product.



(4) When used in an expression, the logical operation is performed after the arithmetic operation, in principle.

(5) The logical operator is used to change the flow of program by use of a condition judging statement, etc. (See IF command in page 91)

EXAMPLE 2

- The following is an example of operation by use of each logical operator.

	Bit Contents																	
# 15 = -16	B15 B0																	
	15	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
	#																	
	-16	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	
63 & 15 = 15	B15 B0																	
	63	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	
	&																	
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
	↓																	
	15	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	
(-1) ! 8 = -1	B15 B0																	
	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	!																	
	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	↓																	
	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
(-1) ≠ 8 = -9	B15 B0																	
	-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
	≠ (\)																	
	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	↓																	
	-9	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	

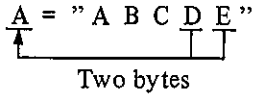
2.5.4 Built-in functions

The built-in function is used to call predetermined operation with respect to operands. The built-in functions are as follows:

Command Name		Function
1	ABF	Absolute value of mathematical expression value (real number)
2	ABS	Absolute value of mathematical expression value (integer)
3	ACOS	Arccosine (\cos^{-1}) of mathematical expression
4	ASIN	Arcsine (\sin^{-1}) of mathematical expression
5	ATAN	Arctangent (\tan^{-1}) of mathematical expression
6	COS	Cosine (cos) of mathematical expression
7	EXP	Value of exponential function which has "e" as a base ($e = 2.718281$)
8	LN	Value of natural logarithm ($\log e$)
9	LOG	Value of common logarithm ($\log 10$)
10	NOT	Generates 1 when mathematical expression value is 0, and 0 when value is not 0.
11	RND	Assigns random number to variable.
12	SIN	Sine (sin) of mathematical expression
13	SQRT	Value of square root of mathematical expression
14	TAN	Tangent (tan) of mathematical expression

2.5.5 Character string operation

The character string operation performs operation regarding the key codes, which correspond to the characters of the last two bytes of the character string, as an integer.



EXAMPLE

```

1 0 0 A = "ABC".....$4243 (key code of "BC") is assigned to variable A.
1 1 0 B = "DEF".....$4546 (key code of "EF") is assigned to variable B.
1 2 0 C = A + B.....$4243 + $4546 is assigned to variable C.
1 3 0 D = "MITSUBISHI".....$4849 (key code of "HI") is assigned to variable D.
1 4 0 E = "ELECTRIC".....$4943 (key code of "IC") is assigned to variable E.
1 5 0 F = D - E.....$ECA9-$D29E is assigned to variable F.
1 6 0 PRINT $A
1 7 0 PRINT $B
1 8 0 PRINT $C
1 9 0 PRINT $D
2 0 0 PRINT $E
2 1 0 PRINT $F
2 2 0 END
OK
> RUN
4 2 4 3.....Content of variable A is displayed in hexadecimal.
4 5 4 6.....Content of variable B is displayed in hexadecimal.
8 7 8 9.....Content of variable C is displayed in hexadecimal.
4 8 4 9.....Content of variable D is displayed in hexadecimal.
4 9 4 3.....Content of variable E is displayed in hexadecimal.
FF 0 6.....Content of variable F is displayed in hexadecimal.
OK
>

```

NOTE

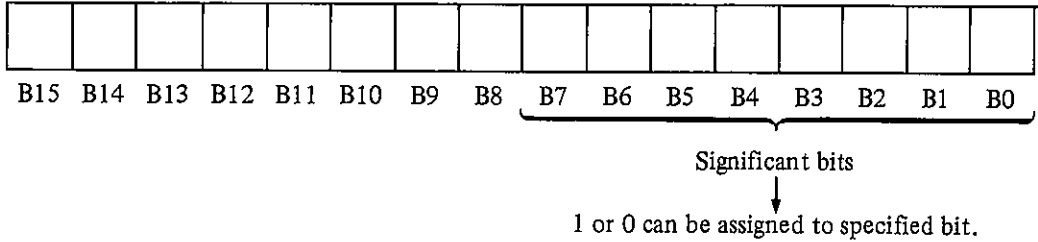
The key codes of characters are as follows:

	GPC-1	GPC-11	KD51
ASCII character	1 byte	1 byte	1 byte
Special symbol	2 bytes	2 bytes	—

*: Special symbol cannot be used for KD51.

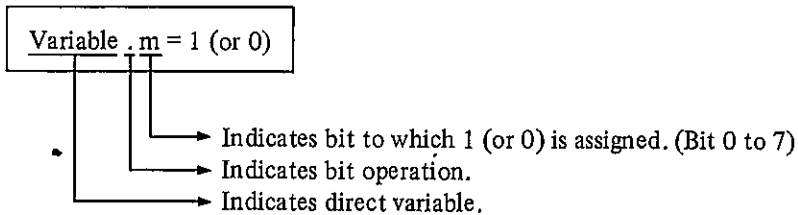
2.5.6 Bit operation

In GPC-BASIC, only integer variables are provided with bit operation command. However, usable bits are 0 to 7 (when there are one or more bytes, the low-order one byte is used). In regards to the setting of eight or more bits, calculation is made by 8's remainder (% 8).



(1) Set and reset of bit

1) Indirect variable



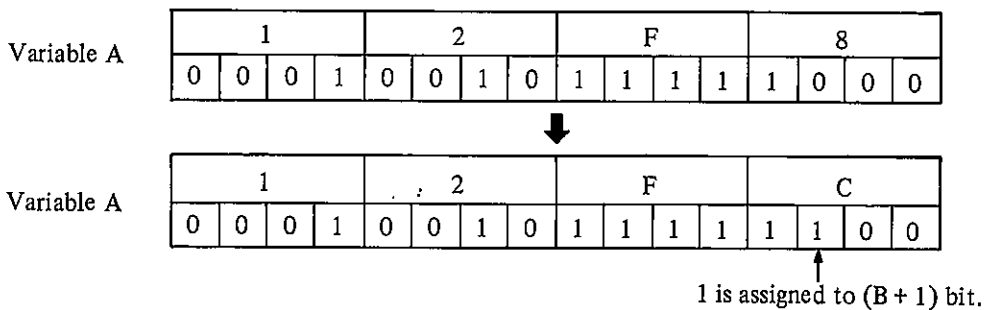
Assign 1 (or 0) to the bit "m" of numeric value stored in the variable.

EXAMPLE

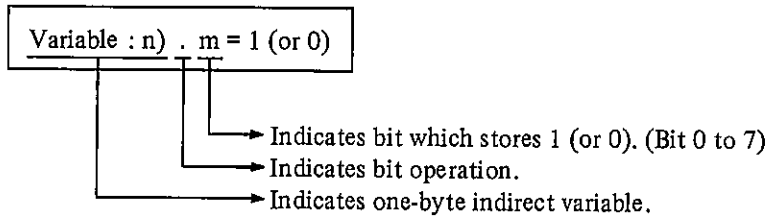
```

1 0 0   A = $ 1 2 F 8 ..... $12F8 is set to variable A.
1 1 0   B = 1 ..... 1 is set to variable B.
1 2 0   A. B + 1 = 1 ..... 1 is assigned to bit (B+1)%8 (second bit) of variable
1 3 0   END ..... A content ($12F8). (Value in variable A turns to $12
                                FC.)

```



2) One-byte indirect variable

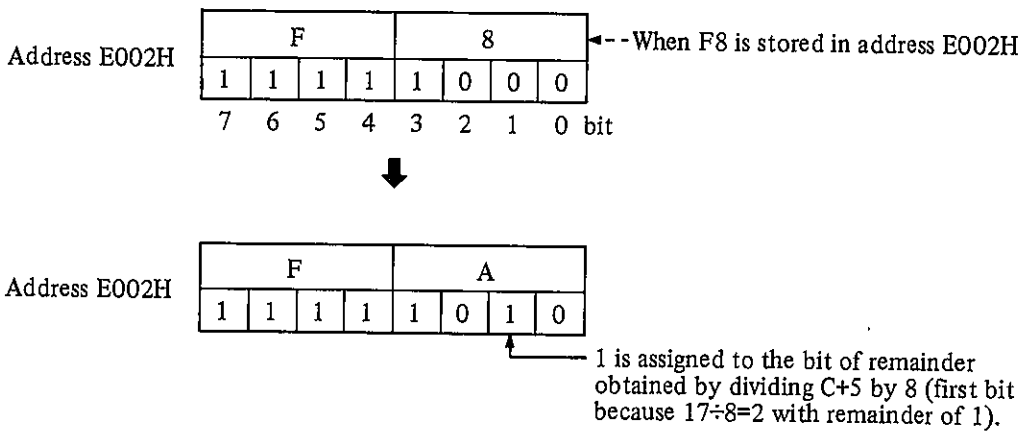


Assign 1 (or 0) to the bit "m" of numeric value stored in the address specified in the variable.

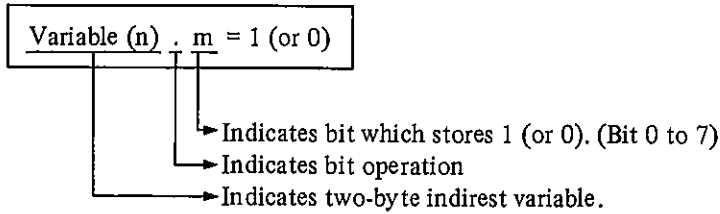
EXAMPLE

```

1 0 0  A = $ E 0 0 0 .....$E000 is set to variable A.
1 1 0  B = 1 .....1 is set to variable B.
1 2 0  C = 1 2 .....12 is set to variable C.
1 3 0  A : B + 1). C + 5 = 1 .....1 is assigned to bit (C+5)%8 (first bit) of address
1 4 0  END .....E000H + (B+1) (address E002H).
    
```



3) Two-byte indirect variable

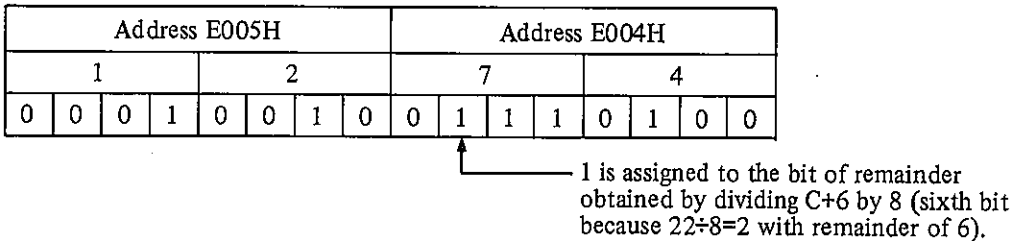
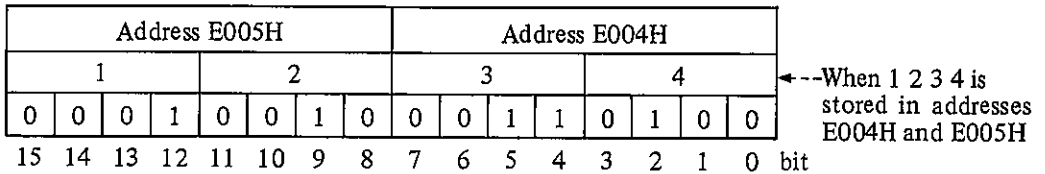


Assign 1 (or 0) to the bit "m" of numeric value stored in the address specified in the variable.

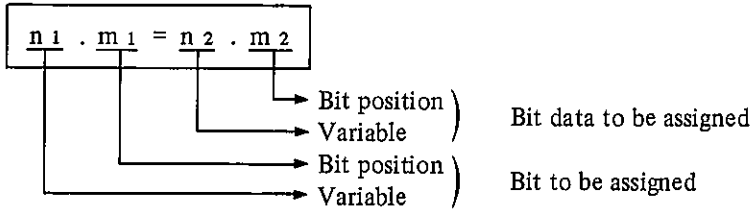
EXAMPLE

```

1 0 0  A = $E 0 0 0 .....$E000 is set to variable A.
1 1 0  B = 1 .....1 is set to variable B.
1 2 0  C = 1 6 .....16 is set to variable C.
1 3 0  A(B+1). C + 6 = 1 .....1 is assigned to bit (C+6)%8 (sixth bit) of address
1 4 0  END .....E000H+(B+1)×2 (address E004H).
    
```



(2) Assignment of bit



Assign the data in the bit "m2" of numeric value stored in the variable n2 to the bit "m1" of numeric value stored in the variable n1.

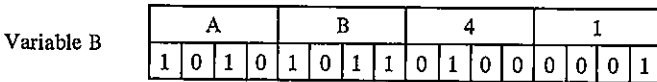
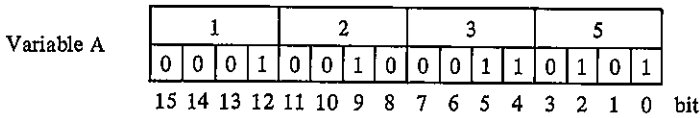
n1 and n2 are direct variables, one-byte indirect variables or two-byte variables.

EXAMPLE 1

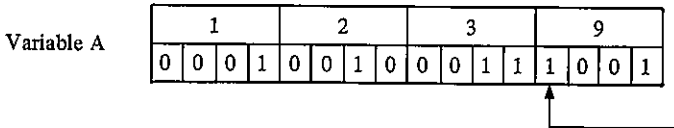
Direct variable

```

1 0 0 A = $ 1 2 3 5 ..... $1235 is set to variable A.
1 1 0 B = $ A B 4 1 ..... $AB41 is set to variable B.
1 2 0 C = 2 ..... 2 is set to variable C.
1 3 0 A . C + 1 = B . 0 ..... Value in bit 0 of variable B is assigned to bit (C+1)
1 4 0 END ..... %8 (third bit) of variable A. Therefore, value of
                                     variable A turns to $1239.
  
```



Value in bit 0 of variable B is assigned to bit (C+1)%8=3 of variable A.



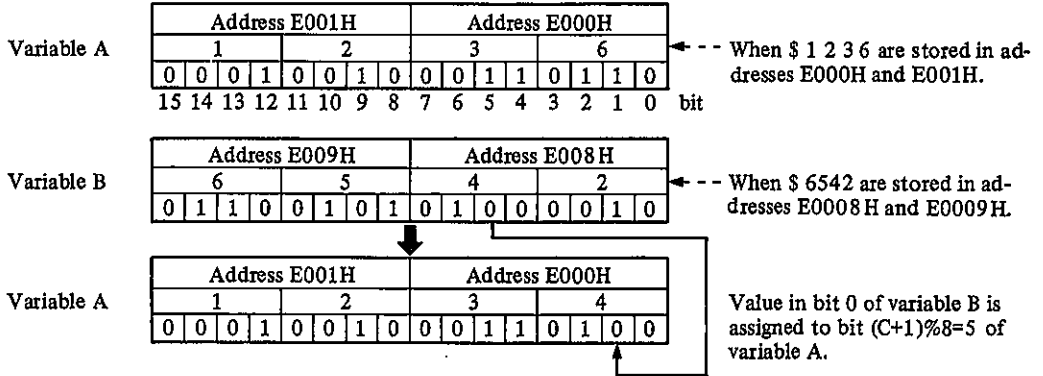
EXAMPLE 2

One-byte indirect variable

```

1 0 0 A = $E 0 0 0 .....$E000 is set to variable A.
1 1 0 B = $E 0 0 6 .....$E006 is set to variable B.
1 2 0 C = 4 .....4 is set to variable C.
1 3 0 A : 0). 1 = B : 2). C + 1 .....Value in bit (C+1)%8 (fifth bit) of address E006H+
1 4 0 END .....1×2 (address E008H) is assigned to the first bit of
address E000.

```



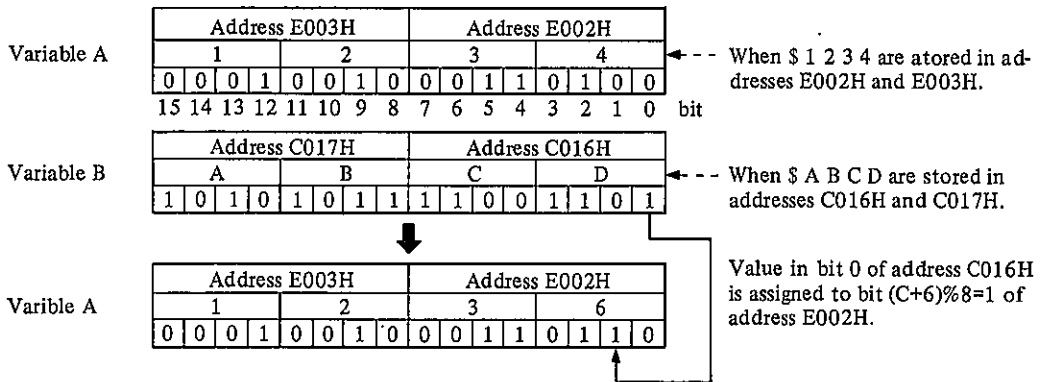
EXAMPLE 3

Two-byte indirect variable

```

1 0 0 A = $E 0 0 0 .....$E000 is set to variable A.
1 1 0 B = $C 0 0 0 .....$C000 is set to variable B,
1 2 0 C = 3 .....3 is set to variable C.
1 3 0 D = 8 .....8 is set to variable D.
1 4 0 A (1). C + 6 = B (D + 3). 0 .....Value in bit 0 of address C000H+(D+3)×2 (address
1 5 0 END .....C016H) is assigned to bit (C+6)%8 (first bit) of address
E000H+1×2 (address E002H).

```



2.5.7 Form of assignment statement

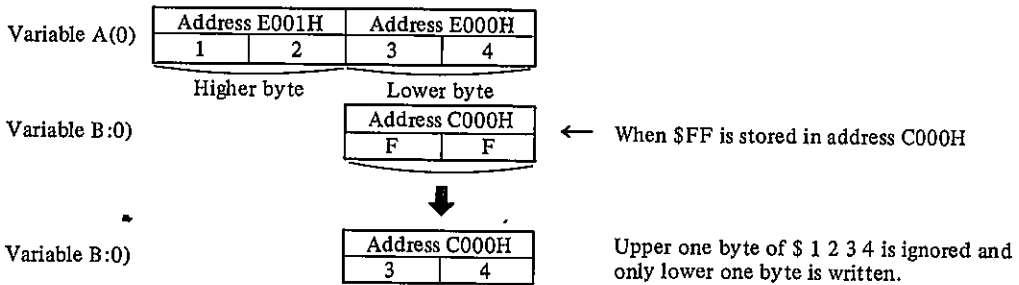
Basically, in regards to the forms of right and left members in an assignment statement, the numbers of bytes in the right and left members must be the same. However, the right and left members can be used as described below.

- (1) Left member (one-byte indirect variable) = right member (two-byte indirect variable, basic variable)
 The low-order one byte of two bytes on the right is assigned to the left member, and the high-order one byte is ignored.

EXAMPLE

```

1 0 0  A = $ E 0 0 0 .....$E000 is set to variable A.
1 1 0  A ( 0 ) = $ 1 2 3 4 .....$1234 is written to addresses E000H and E001H.
1 2 0  B = $ C 0 0 0 .....$C000 is set to variable B.
1 3 0  B : 0 ) = A ( 0 ) .....Content of address E000H ($34) is written to address
1 4 0  END .....C000H. (Content of address E001H ($12) is ignored.)
    
```



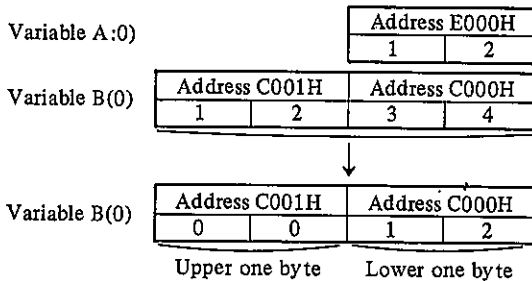
- (2) Left member (two-byte indirect variable, basic variable) = right member (one-byte indirect variable)
 The right member is assigned to the low-order one byte on the left, and 0 is written to the high-order one byte.

EXAMPLE

```

1 0 0  A = $ E 0 0 0 .....$E000 is set to variable A.
1 1 0  A : 0 ) = $ 1 2 .....$12 is written to address E000H.
1 2 0  B = $ C 0 0 0 .....$C000 is set to variable B.
1 3 0  B ( 0 ) = A ( 0 ) .....Content of address E000H ($12) is written to addresses
1 4 0  END .....C000H and C001H.
    
```

Assuming that \$ 1 2 3 4 is stored in addresses C000H and C001H



(3) Left member (two-byte indirect variable, basic variable) = right member (four-byte indirect variable)

Not applicable to KGPC1

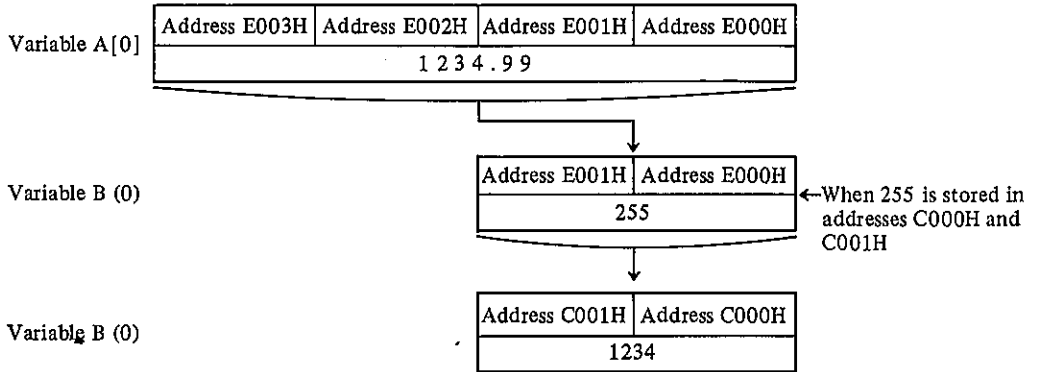
The floating-point type real number on the right is converted into an integer type and assigned to the left member. Numeric values usable as variables are -32767 to 32767.

EXAMPLE

```

1 1 0 A = $ E 0 0 0 ..... $E000 is set to variable A.
1 1 0 B = $ C 0 0 0 ..... $C000 is set to variable B.
1 2 0 A[0] = 1 2 3 4 . 9 9 ..... 1234.99 is written to addresses E000H to E003H.
1 3 0 B(0) = A[0] ..... Content of addresses E000H to E003H (1234.99) is
1 4 0 END ..... converted into integer form (1234) and assigned to
                           addresses C000H and C001H.

```



(4) Left member (four-byte indirect variable) = right member (two-byte indirect variable)

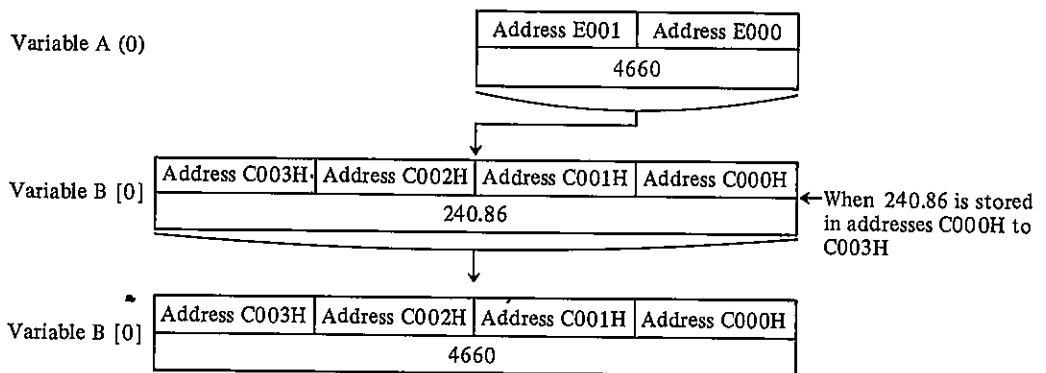
Not applicable to KGPC1

The two-byte indirect variable on the right is converted into a floating-point type real number and assigned to the left member.

EXAMPLE

```

1 0 0  A = $E 0 0 0 .....$E000 is set to variable A.
1 1 0  B = $C 0 0 0 .....$C000 is set to variable B.
1 2 0  A ( 0 ) = 4 6 6 0 .....4660 is written to addresses E000H and E001H.
1 3 0  B ( 0 ) = A ( 0 ) .....Content of addresses E000H and E001H (4660) is
1 4 0  END .....converted into floating-point type real number and
                           .....written to addresses C000H to C003H.
  
```

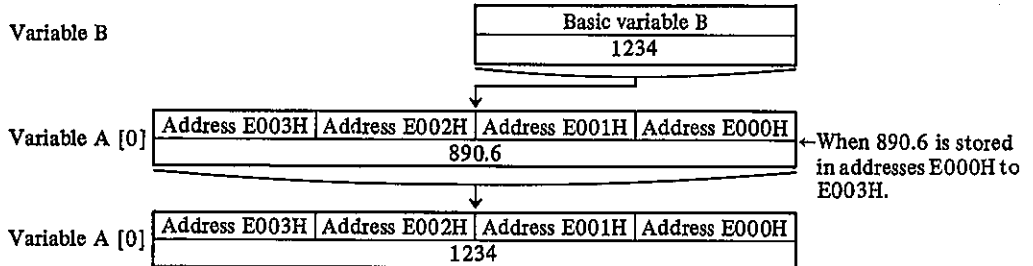


- (5) Left member (four-byte indirect variable) = right member (two-byte basic variable)
 The integer type basic variable on the right is converted into a floating-point type real number and assigned to the left member.

EXAMPLE

```

1 0 0 A=$E000 .....$E000 is set to variable A.
1 1 0 B=1234 .....1234 is written to variable B.
1 2 0 A[0]=B .....Content of variable B (1234) is converted into
1 4 0 END .....floating-point type real number and written to
                  addresses E000H to E003H.
  
```



CAUTION

When a four-byte indirect variable is on the left, a one-byte indirect variable cannot be used on the right.

`A [0] = A:0) One-byte indirect variable on the right`

2.5.8 Comparison of expressions

- (1) Comparison statement

In an integer expression, a compare statement can be used. When the result of "n1 (comparing operator) n2" is true, the value is 1. When the result is false, the value is 0.

EXAMPLE 1

When B=10, C=12, D=14, E=16,

$$\begin{aligned}
 A &= (B > C) + (D < E) \\
 &= 0 + 1 \\
 &= 1
 \end{aligned}$$

CAUTION

In a real expression (four-byte indirect variable), the compare statement cannot be used.

(2) Comparison in IF statement

In an IF statement, the compare statement can be used for both the integer expression and real expression. For the integer expression, be sure to provide a period behind IF.

IF (A > B) = (C > D)Integer expression

→ When (A > B), (C > D) and (A < B), (C < D), (Left member) = (right member)

IF. A[m1] = B[m2]Real expression

EXAMPLE 1

For integer expression

IF (A > B) = (C > D) GO TO **

When the value of (A > B) and the value of (C > D) are not equal, execution goes to the line number **

EXAMPLE 2

For real expression

```
100 A=$E000
110 B=$EF00
120 IF. A [0] = B [2] GOTO * *
130 END
```

When the contents of addresses E000H to E003H (A [0]) and those of addresses EF08H to EFOBH (B [2]) are equal, execution goes to the line number * *

2.6 How to Prepare Program

2.6.1 How to prepare program

The standard operating procedure for preparing a new program with GPC-BASIC is shown below.

```
OK
>■
```

Set "BASIC PROGRAMMING" at initial setting to display the left figure on the screen.



```
OK
>NEW [CR]
OK
>■
```

Press **N E W [CR]** keys.
(The BASIC program area is initialized to be ready for preparing a new program.)



```
OK
>NEW
OK
>AUTO [CR]
100 ■
```

Press **A U T O [CR]** keys.
(Line numbers are automatically allotted to the heads of lines.)



```
OK
>NEW
OK
>AUTO
100 REM "HOW TO PREPARE PROGRAM" [CR]
110
.
.
.
800 END [CR]
810 [ESC]
OK
>■
```

Prepare the program.
When one line has been entered, press the **[CR]** key and prepare the next line. When the last line has been entered, press the **[ESC]** key to end the execution of AUTO command.

NOTE
Program which can be prepared in one line is a maximum of 72 bytes.



```
OK
>NEW
OK
>AUTO
100 REM "HOW TO PREPARE PROGRAM" [CR]
110
.
.
.
800 END
810
OK
>BYE [CR]
```

When it is desired to return to the initial setting screen after the completion of program entry and correction, press the **[BYE]** keys.
When it is desired to suspend the preparation of program and display another screen or before turning off the power, be sure to also press the **[BYE]** keys.

2.6.2 Correction of program

(1) Error is found in line which is being prepared

- 1) Erase characters one by one by use of **DEL** key up to the error portion and then enter data again through keys.

EXAMPLE

```
1 8 0 CIRCLE(300,200) ■
```

A period (.) has been entered instead of a comma (,) by mistake.



```
1 8 0 CIRCLE(300 ■
```

Press the **DEL** key five times to return the cursor to the position of period (.)



```
1 8 0 CIRCLE(300,200),150 ■
```

Enter the comma (,) and following data again.

- 2) Press the **CAN** key to erase all of one line, and enter data anew, beginning with the line number.

EXAMPLE

```
1 8 0 CIRCLE(300,200),150 ■
```

The CIRCLE command has been used instead of the LINE command by mistake.



```
■
```

Press the **CAN** key to erase all of one line.



```
1 8 0 LINE(0,0)-(639,399)
```

Enter data again beginning with the line number.

- (2) To correct a line which has been entered
By EDIT command, display the line to be corrected and make correction.

EXAMPLE

```
OK
>EDIT 180
180 ■CERCLE(300-200),150
```

Press the **EDIT** **SPACE** **180** **CR** keys. Enter the line number, which is desired to be called, after EDIT.



```
OK
>EDIT 180
180 CERCLE(300-200),150
```

To correct CERCLE to CIRCLE

Press the **SPACE** key twice to move the cursor to the position of "E".



```
OK
>EDIT 180
180 CERCLE(300-200),150
```

After pressing the **C** key to select the replace mode, press the **I** key to correct the spell.



```
OK
>EDIT 180
180 CIRCLE(300-200),150
```

To correct - (hyphen) to ,

After pressing the **ESC** key to clear the replace mode, press the **SPACE** key nine times to move the cursor to the position of hyphen (-).



```
OK
>EDIT 180
180 CIRCLE(300,200),150
```

After pressing the **C** key to select the replace mode, press the **,** key to correct to a correct form.



```
OK
>EDIT 180
180 CIRCLE(300,200),150
OK
>■
```

Press the **CR** key to clear the execution of EDIT command.

(3) Addition, insertion, rewriting and deletion of program line

1) Addition of program

When adding a program, specify the line number of program, which is added, to a line number located next to the last line number of program to which the new program is added.

```
OK
>LIST 
 100 REM "ADDITION OF PROGRAM "
 110
  :
  :
  :
 340 GOTO 180
OK
>AUTO 350 
350 CIRCLE(320,200),150,F 
360 LINE(0,0)-(320,200) 
370 
OK
>LIST 
 100 REM "ADDITION OF PROGRAM "
 110
  :
  :
  :
 340 GOTO 180
 350 CIRCLE(320,200),150,F
 360 LINE(0,0)-(320,200)
OK
>
```

Add a program of line numbers 350 and 360 to below the program of line numbers 100 to 340.

} Display after addition

2) Insertion of program

When inserting a line into a program, specify the line number of program, which is inserted, between line numbers where the new line is inserted.

```
OK
>LIST
 100 REM "INSERTION OF PROGRAM "
 110 FOR I=0 TO100
 120 A=I*I
 130 NEXT I
 140 END
```

```
OK
>125 PRINT A[CR]
```

```
OK
```

```
>LIST[CR]
 100 REM "INSERTION OF PROGRAM "
 110 FOR I=0 TO100
 120 A=I*I
 125 PRINT A
 130 NEXT I
 140 END
```

```
OK
```

```
>
```

Insert a program of line number 125 between line numbers 120 and 130.

} Display after insertion of line number 125

3) Rewriting of program

When rewriting one line of program to another contents, specify the line number of program to be rewritten at the line number of program which is rewritten.

```
OK
>LIST[CR]
100 GCOLOR(1)
110 FTYPE(1)
120 FSTYLE(3)
130 CIRCLE(320,200),150,F
140 END
OK
>130 LINE(50,50)-(500,200),BF[CR]
OK
>LIST[CR]
100 GCOLOR(1)
110 FTYPE(1)
120 FSTYLE(3)
130 LINE(50,50)-(500,200),BF
140 END
OK
>
```

Rewrite the CIRCLE command of line number 130 to the LINE command.

} Display after rewriting of line number 130

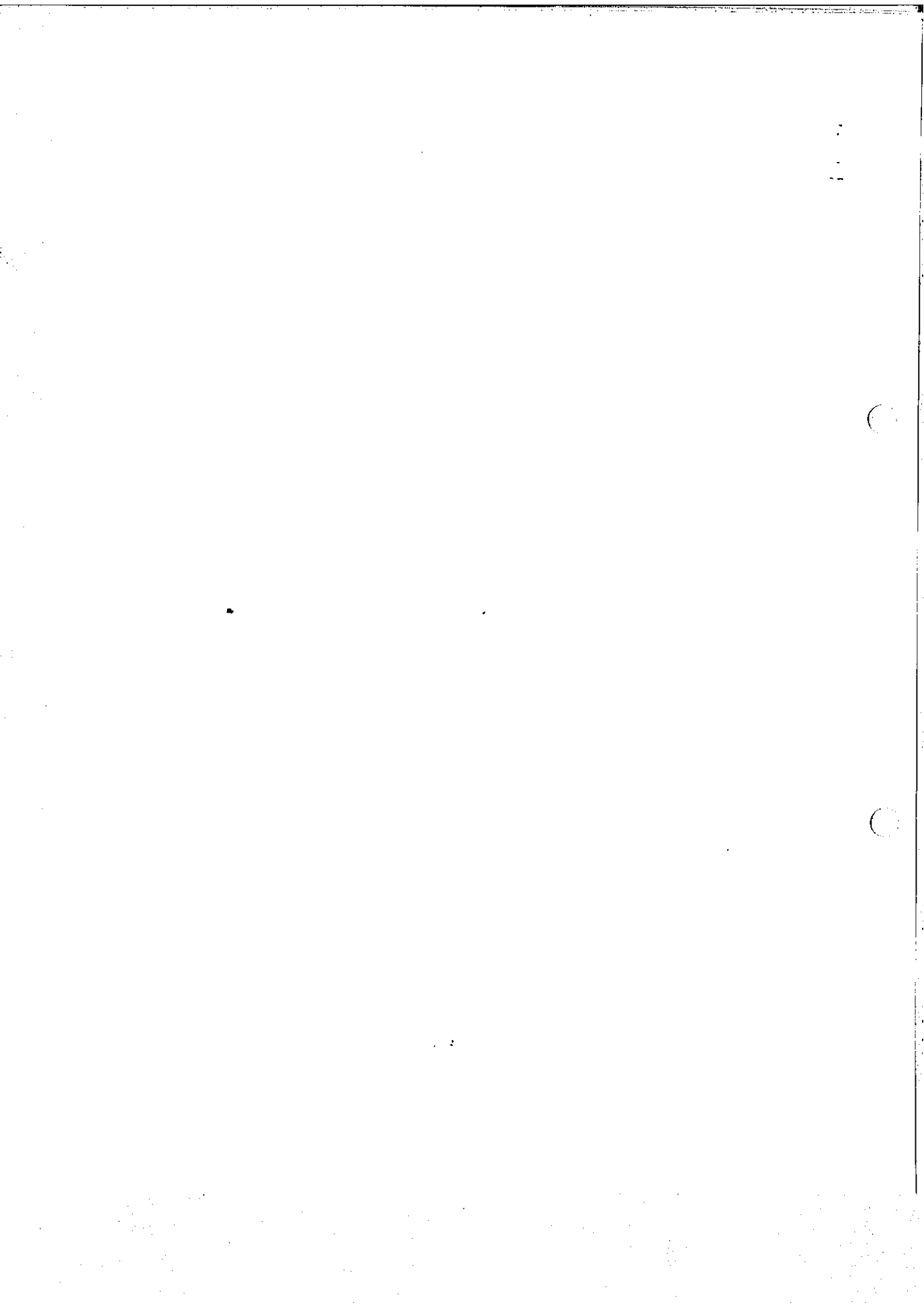
4) Deletion of program

When deleting one line of program, specify the line number of line which is desired to be deleted, and then press the [CR] key.

```
OK
>LIST[CR]
100 GCOLOR(1)
110 FTYPE(1)
120 FSTYLE(3)
130 CIRCLE(320,200),150,F
140 END
OK
>120 [CR]
OK
>LIST[CR]
100 GCOLOR(1)
110 FTYPE(1)
130 CIRCLE(320,200),150,F
OK
>
```

Delete line number 120.

} Display after deletion of line number 120



A decorative rectangular border with a repeating floral or scrollwork pattern surrounds the title text.

GPC-BASIC COMMANDS

3. GPC-BASIC COMMANDS

3.1 Keyboard Commands

The keyboard command is an instruction which is entered without a line number. This command is executed immediately after it is entered and the instruction does not remain in the memory. The keyboard commands include the following commands:

Command Name	Reference Page
AUTO	57
BYE	58
CONT	59
COMPILE	60
DELETE	61
EDIT	62
EXECUTE	65
LIST	66
LLIST	69
NEW	70
RENUM	71
RUN	73
ZDV	74
LINE DELETE	75

AUTO

KEY

FUNCTION

Gives a line number automatically at the head of line of program.

FORM

AUTO [line number[, increment]]

EXPLANATION

- The value indicated in [line number] is the first line number. Depression of the **CR** key at the end of this line automatically displays the line number to which the value indicated in [increment] is added.
- When [line number] and [increment] are omitted, the first line number is set to 100 and the increment is set to 10.
- When only [line number] is specified, the first line number is set to the specified value and the increment is set to 10.
- To end the ATUO command, press the **ESC** key.
- The line number is an integer ranging from 1 to 32767.

EXAMPLE 1

[line number] = 500 and [increment] = 20 are set

```
OK
>AUTO 500,20
 500 REM "AUTO Test of AUTO command."
 520 REM
 540
```

EXAMPLE 2

Only [line number] = 200 is set

```
OK
>AUTO 200
 200
 210
 220
```



Since the increment is omitted, 10 is added to line number each time.

EXAMPLE 3

[line number] and [increment] are omitted

```
OK
>AUTO
 100
 110
 120
```



Since the line number and increment are omitted, the first line number is 100, and 10 is added each time.

BYE

KEY

FUNCTION

By using this command after BASIC programming is completed, control is returned to the OS side and the initial setting screen of BASIC programming is restored.

FORM

BYE

EXPLANATION

- Use this command when it is desired to restore the initial setting screen of BASIC programming.
- By executing the BYE command, the additional program head address is registered in the work area. Therefore, be sure to execute the BYE command after the completion of programming.

CONT

FUNCTION

Executes a program, which is at stop due to the execution of the BREAK statement (see p.78), beginning with the next line number.

FORM

CONT

EXPLANATION

- After interrupting the execution of program by the BREAK command, the run of program can be resumed, beginning with the line number located next to the line number provided with the BREAK command, by the CONT command.
- The CONT command is normally used during debugging. The CONT command allows the check or modification of intermediate result by use of command and mode statement when the execution of program is stopped.

EXAMPLE

Continuation of program by CONT command

```

OK
>LIST
 100 REM "EXERCISE"
 110 PRINT "BREAK" ..... "BREAK" is displayed on the screen.
 120 BREAK ..... Execution is interrupted.
 130 PRINT "CONTINUE" ..... After "CONT" is pressed, "CONTINUE" is displayed
 140 STOP ..... on the screen.
OK
>RUN ..... Program is run.
BAEAK ..... Display by line number 120.
OK
>CONT ..... Press "CONT".
CONTINUE ..... Display by line number 130.

```


COMPILE

KEY

FUNCTION

Checks the grammar of BASIC program and converts the format of program into the final format.

FORM

COMPILE

EXPLANATION

- Converts a program into one which can run multi task.
- Checks the grammar of only GOTO statement (see P.89) and GOSUB statement (see P.90).
- After preparing or modifying a program, be sure to convert the text to the final format by the COMPILE command or RUN command. Note that if a program, which has not been converted into the final format, is executed by multi task or EXECUTE command (see P.65), the program cannot be executed normally.

DELETE

FUNCTION

Deletes a program from the specified line number to the specified line number.

FORM

DELETE line number 1, line number 2

EXPLANATION

- Deletes all lines of line number 1 through line number 2.
- When either of line number 1 or 2 does not exist, deletion is not made.

EXAMPLE

Deletion of program by DELETE command

```

OK
>LIST
 100 REM "EXERCISE"
 110 INPUT "X= ", X
 120 INPUT "Y= ", Y
 130 Z=X+Y
 140 PRINT "X+Y= ", Z
 150 END
} .....List before execution of DELETE.

OK
>DELETE 110,140 .....Line numbers 110 to 140 are deleted.
OK
>LIST
 100 REM "EXERCISE"
 150 END
} .....List after execution of DELETE.

```

EDIT

KEY

FUNCTION

Corrects a statement in one line.

FORM

EDIT line number

EXPLANATION

- Corrects the line specified by a line number.
- When the correction of line, which has been called by the EDIT command, is all completed, restore the command accept state by the **CR** key.
- There are correction modes by use of the **SPACE** key, **DEL** key, **L** key, **X** key, **I** key, **D** key, **Q** key, and **C** key. To cancel each correction mode, press the **ESC** key.

EXAMPLE

```
100 FOR I=0 TO 150
110 X=I*I
120 PRINT X
130 NEXT I
140 END
```

When it is desired to correct the line number 100 of above program, press the following keys:

E D I T 1 0 0 CR

```
OK
>EDIT 100
100_FOR I=0 TO 150
```

The underline indicates the cursor position.

1. Moving the cursor

Move the cursor up to the character which you desire to correct.

SPACE key

Moves the cursor rightward to the end of line.

DEL key

Moves the cursor leftward to the line number.

[L] key

Moves the cursor to the initial position.

```
OK
>EDIT 100
100 FOR I=0 TO 150
```

When **[L]** key is pressed

```
OK
>EDIT 100
100_ FOR I=0 TO 150
```

[X] key

Moves the cursor to the end of line. Used to newly insert a statement.

```
OK
>EDIT 100
100_ FOR I=0 TO 150
```

When **[X]** key is pressed

```
OK
>EDIT 100
100 FOR I=0 TO 150_
```

When **[SPACE]** **[S]** **[T]** **[E]** **[P]** **[SPACE]** **[Z]** keys are pressed

```
OK
>EDIT 100
100 FOR I=0 TO 150 STEP 2_
```

2. Insert mode (**[I]** key)

Adds a character between the cursor position and its left side character.

```
OK
>EDIT 100
100 FOR I=0 TO 150 STEP 2
```

When **[I]****[I]** keys are pressed

```
OK
>EDIT 100
100 FOR I=10 TO 150 STEP 2
```

3. Replace mode (**C** key)

Replaces the character, where the cursor is located, with a character which is entered through the key.

```
OK
>EDIT 100
100 FOR I=0 TO 150 STEP 2
```

When **C** **2** **5** keys are pressed

```
OK
>EDIT 100
100 FOR I=0 TO 125 STEP 2
```

4. Delete mode

D key

Deletes a character where the cursor is located.

```
OK
>EDIT 100
100 FOR I=10 TO 125 STEP 2
```

When **D** key is pressed

```
OK
>EDIT 100
100 FOR I=10 TO 25 STEP 2
```

After pressing the **C** key, **I** key and **X** key, press the **DEL** key. The character in front of the cursor is deleted.

```
OK
>EDIT 100
100 FOR I=10 TO 125 STEP 2
```

When **C** **DEL** (or **I** **DEL**, **X** **DEL**) keys are pressed.

```
OK
>EDIT 100
100 FOR I=10 TO 25 STEP 2
```

5. Cancel mode (**Q** key)

Cancels all corrections that you have made and ends the correction mode.

EXECUTE

FUNCTION

Executes program without the check of BASIC program grammar or the conversion of a program into the final format.

FORM

```
EXECUTE [line number]
```

EXPLANATION

- Executes a program without the conversion into the final format, which can run multi task, or the check of grammar of GOTO statement (see P.89) or GOSUB statement (see P.90).
- Before executing the EXECUTE command, be sure to execute the RUN command or COMPILE command one or more times to convert the program into the final format.
- Note that if a program, which is not in the final format, is executed by the EXECUTE command, the program cannot be normally executed.

LIST

TYPE	A
------	---

KEY

FUNCTION

Displays all or part of programs, which are located within the memory, on the screen.

FORM

LIST [[line number 1] [- [line number 2]]]

EXPLANATION

- A period (.) can be used instead of the LIST command.
- A program of up to 17 lines is displayed on the CRT.
- When the setting of display is 18 line or more, 17 lines are displayed beginning with the line number specified in [line number 1], then the lines are scrolled one by one, and up to the line number specified in [line number 2] is displayed.
- When it is desired to interrupt the display midway, press the **CAN** key. When it is desired to resume the scrolling operation, press any key other than the **CONT** / **C** key.
- When only [line number 1] is specified, only the content of specified line number is displayed.
- When [line number 1-] is specified, the contents of the specified line number through the final line number are displayed.
- When [-line number 2] is specified, the contents of the first line number through the specified line number are displayed.
- When [line number 1 - line number 2] is specified, the contents of [line number 1] to [line number 2] are displayed.
- When line numbers are not specified, all programs within the memory are displayed.
- When the specified line number does not exist, a line number greater than and the closest to the specified line number is selected for [line number 1], and a line number less than and the closest to the specified line number, is selected for [line number 2].
- When [line number 1] is greater than or equal to [line number 2], only the content of [line number 1] is displayed.
- When it is desired to end the display midway, press the **CONT** / **C** key.

EXAMPLE

LIST.....	All lines are displayed.
LIST 500.....	Line of line number 500 is displayed.
LIST 150-.....	Line of line number 150 through the last line are displayed.
LIST -1000.....	The first line through line of line number 1000 are displayed.
LIST 150-1000.....	Line of line number 150 through line of line number 1000 are displayed.

LIST

TYPE | B

KEY

FUNCTION

Displays all or part of programs, which are located within the memory, on the screen.

FORM

LIST [[line number 1] [-[line number 2]]]

EXPLANATION

- A period (.) can be used instead of the LIST command.
- A program of up to 20 lines is displayed on the CRT.
- A program is displayed and interrupted in units of 10 lines. By pressing any key other than the **CONT** / **C** key, a program of 10 lines, beginning with a line number next to the presently displayed program, is displayed.
- When only [line number 1] is specified, only the content of specified line number is displayed.
- When [line number 1 -] is specified, the contents of the specified line number through the final line number are displayed in units of 10 lines.
- When [- line number 2] is specified, the contents of the first line number through the specified line number are displayed in units of 10 lines.
- When [line number 1 - line number 2], is specified, the contents of [line number 1] through [line number 2] are displayed in units of 10 lines.
- When line numbers are not specified, all programs within the memory are displayed in units of 10 lines.
- When the specified line number does not exist, a line number greater than and the closest to the specified line number is selected for [line number 1], and a line number less than and the closest to the specified line number, is selected for [line number 2].
- When [line number 1] is greater than or equal to [line number 2], only the content of [line number 1] is displayed.
- When it is desired to end the display midway, press the **CONT** / **C** key.

EXAMPLE

LIST	All lines are displayed.
LIST 500	Line of line number 500 is displayed.
LIST 150-	Line of line number 150 through the last line are displayed.
LIST -1000	The first line through line of line number 1000 are displayed.
LIST 150-1000	Line of line number 150 through line of line number 1000 are displayd.

LIST

TYPE C

KEY

FUNCTION

Displays all or part of programs, which are located within the memory, on the screen.

FORM

LIST [[line number 1] [– [line number 2]]]

EXPLANATION

- A period (.) can be used instead of the LIST command.
- A program of up to 20 lines is displayed on the CRT.
- A program is displayed and interrupted in units of 20 lines. By pressing any key other than the **CONT** / **C** key, a program of 20 lines, beginning with a line number next to the presently displayed program, is displayed.
- When only [line number 1] is specified, only the content of specified line number is displayed.
- When [line number 1 –] is specified, the contents of the specified line number to the final line number are displayed in units of 20 lines.
- When [– line number 2] is specified, the contents of the first line number to the specified line number are displayed in units of 20 lines.
- When [line number 1 – line number 2] is specified, the contents of [line number 1] to [line number 2] are displayed in units of 20 lines.
- When line numbers are not specified, all programs within the memory are displayed in units of 20 lines.
- When the specified line number does not exist, a line number greater than and the closest to the specified line number is selected for [line number 1], and a line number less than and the closest to the specified line number, is selected for [line number 2].
- When [line number 1] is greater than or equal to [line number 2], only the content of [line number 1] is displayed.
- When it is desired to end the display midway, press the **CONT** / **C** key.

EXAMPLE

```
LIST .....All lines are displayed.
LIST 500 .....Line of line number 500 is displayed.
LIST 150 - .....Line of line number 150 through the last line are
                displayed.
LIST -1000 .....The first line through line of line number 1000 are
                displayed.
LIST 150 -1000 .....Line of line number 150 through line of line number
                1000 are displayed.
```

LLIST

FUNCTION

Prints all or part of programs, which are located within the memory, on the screen.

FORM

LLIST [[line number 1] [- [line number 2]]]

EXPLANATION

- When only [line number 1] is specified, only the content of specified line number is printed.
- When [line number 1 -] is specified, the contents of the specified line number to the final line number are printed.
- When [- line number 2] is specified, the contents of the first line number to the specified line number are printed.
- When [line number 1 - line number 2] is specified, the contents of [line number 1] to [line number 2] are printed.
- When line numbers are not specified, all programs within the memory are printed.
- When [line number 1] is greater than or equal to [line number 2], only the content of [line number 1] is printed.
- When the specified line number does not exist, a line number greater than and the closest to the specified line number is selected for [line number 1], and a line number less than and the closest to the specified line number, is selected for [line number 2].
- When the printing is completed, OK is displayed on the screen.
- When it is desired to end the printing midway, press the **CONT** / **C** key. However, the printing does not stop immediately since the printing is done up to the portion which has been sent to the printer.

NEW

FUNCTION

Deletes programs within the memory.

FORM

NEW

EXPLANATION

- The NEW command initializes the memory before a new program is entered.
- If the NEW command is executed, variables are not initialized.
- When the NEW command is executed, the range set by the BASIC programming initial setting screen is initialized and then BASIC returns to the command accept state.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 INPUT A
120 X=A*A
130 PRINT X
140 END
```

```
OK
>NEW.....Program is deleted.
OK
>LIST.....Since all program has been deleted by the NEW
OK.....command, nothing is displayed if the LIST command
>.....is executed.
```

RENUM

TYPE	A
------	---

KEY

FUNCTION

Renumbers the lines of program.

FORM

RENUM

EXPLANATION

- Renumbers the lines of program in increments of 10, beginning with 100.
- The RENUM command also changes line numbers, which are specified by the GOTO (see P.89) or GOSUB (see P.90) command, according to new line numbers.
- If a line number, which does not exist in the GOTO or GOSUB statement, is specified, error message is displayed, and the RENUM command is not executed.
- The RENUM command cannot generate a line number which exceeds 32767.

EXAMPLE

```
OK
>LIST
  90 REM "EXERCISE"
 100 CLS
 105 INPUT A, B, C
 110 X=A+B+C
 118 Y=A*A+B*B+C*C
 130 Z=X/3
 150 PRINT "A=", A, "B=", B, "C=", C
 153 PRINT "X=", X,
 158 PRINT "Y=", Y
 160 PRINT "Z=", Z
 170 GO TO 100
```

Program before execution of RENUM.

```
OK
>RENUM.....Lines are renumbered in increments of 10, beginning
with line number 100.
```

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 INPUT A, B, C
 130 X=A+B+C
 140 Y=A*A+B*B+C*C
 150 Z=X/3
 160 PRINT "A=", A, "B=", B, "C=", C
 170 PRINT "X=", X
 180 PRINT "Y=", Y
 190 PRINT "Z=", Z
 200 GO TO 110
```

Program after execution of RENUM.

```
OK
>
```

RENUM

TYPE B

KEY

FUNCTION

Renumbers the lines of program.

FORM

RENUM [new starting line number[,old starting line number] [,increment]]

EXPLANATION

- The lines of program, which begins with the line number specified in [old starting line number], are renumbered by increasing the numbers by the values specified in [increment], beginning with the line number specified by [new starting line number].
- When [new starting line number] is omitted, the new line number begins at 100.
- When [old starting line number] is omitted, the new line number begins with the lowest line number of old line numbers. When omitting [old starting line number], specify [new starting line number,increment].
- When [increment] is omitted, line numbers are increased by 10s.
- The RENUM command also changes a line number, which is specified by the GOTO or GOSUB statement, according to new line numbers.
- When a line number specified in [old starting line number] does not exist, error message is displayed.
- The RENUM command cannot generate a line number which exceeds 32767.

EXAMPLE

```
OK
>LIST
 180 REM "EXERCISE"
 190 CLS
 200 X=1
 210 FOR A=1 TO 9
 215 B=X*A
 218 PRINT B,
 220 NEXT A
 223 PRINT
 224 IF X<9 X=X+1; GOTO210
 230 END
```

.....Program before execution of RENUM.

```
OK
>RENUM 500, 180, 20.....Lines of program beginning with line number 180 are
renumbered in increments of 20, beginning with new
line number 500.
```

```
OK
>LIST
 500 REM "EXERCISE"
 520 CLS
 540 X=1
 560 FOR A=1 TO 9
 580 B=X*A
 600 PRINT B,
 620 NEXT A
 640 PRINT
 660 IF X<9 X=X+1; GOTO560
 680 END
```

.....Program after execution of RENUM.

RUN

FUNCTION

Runs a program.

FORM

RUN [line number]

EXPLANATION

- Executes a program, beginning with the line number specified in [line number].
- When [line number] is omitted, the program is run, beginning with the lowest line number.
- The RUN command converts a program into the final format which can run multi task, and checks the grammar of BASIC program.
- Since the RUN command initiates run after calculating the address of jump destination for GOTO (see P.89) or GOSUB (see P.90), it takes some time until the user program is actually run.

EXAMPLE

OK
> RUNProgram is run, beginning with the lowest line number.

OK
> RUN 500Program is run, beginning with line number 500.

ZDV

KEY

FUNCTION

Displays the I/O consoles which are presently specified.

FORM

ZDV

EXPLANATION

- Displays the device numbers of I/O consoles, which are specified by the ZIDV (see P.142) and ZODV (see P.147), at the output console specified by ZODV.

EXAMPLE

```
OK  
> ZDV  
0 0 0 0  
OK  
>
```

Output device number is displayed.

Input device number is displayed.

NOTE

For I/O consoles which correspond to I/O device numbers, see the ZIDV and ZODV commands.

LINE DELETE

FUNCTION

Deletes one line of program.

FORM

Line number **CR** key

EXPLANATION

- Deletes the specified line number.

EXAMPLE

```

OK
>LIST
 100 REM "EXERCISE"
 110 INPUT "X=", X
 120 INPUT "Y=", Y
 130 Z=X+Y
 140 PRINT X, "+", Y, "=", Z
 150 END

```

.....Before execution of command.

```

OK
>110 CR Key.....Line number 110 is deleted.
OK
>120 CR Key.....Line number 120 is deleted.
OK
>130 CR Key.....Line number 130 is deleted.
OK
>140 CR Key.....Line number 140 is deleted.
OK
>LIST
 100 REM "EXERCISE"
 150 END

```

..... After execution of command.

```

OK
>

```


3.2 Program Commands

- The program command performs basic processing and action or controls program flow.
- Program commands include commands which are functions and which are not functions.
- There are the following program commands:

Program Command			
Those Which Are Not Functions			Functions
Command Name	Command Name	Command Name	Command Name
BREAK	GOSUB · · · RETURN	ONGOSUB	CALL
COLOR	IF	ONGOTO	INKEY
CLS	INPUT	OPEN	PEEK
CLOSE	LCOPY	POKE	SIZE
END	LET	PRINT	
FOR · · · NEXT	LOCATE	REM	
GOTO	LPRINT	STOP	

3.2.1 Program commands which are not functions

- The formats of program commands, which are not functions, are as follows:

```
100 CLS
110 COLOR1
```

- There are the following program commands which are not functions:

Command Name	Reference Page	Command Name	Reference Page	Command Name	Reference Page
BRAKE	78	GOSUB... RETURN	90	ONGOSUB	106
COLOR	79	IF	91	ONGOTO	107
CLS	81	INPUT	93	OPEN	108
CLOSE	83	LCOPY	96	POKE	117
END	86	LET	98	PRINT	118
FOR...NEXT	87	LOCATE	99	REM	124
GOTO	89	LPRINT	103	STOP	125

BREAK

FUNCTION

Interrupts the run of program and restores the command accept state.

FORM

BREAK

EXPLANATION

- The BREAK command may be used anywhere in a program.
- By entering the CONT command (see P.59) after the run of program is interrupted and the command accept state is restored by the execution of BREAK command, a program can be run, beginning with the line number following the BREAK command.
- The BREAK command cannot be used for multi task.
- When a semicolon (;) is provided behind the BREAK command and the next statement is written behind the semicolon, the statement located behind the BREAK command is executed.

EXAMPLE 1

```

OK
>LIST
 100 REM "EXERCISE"
 110 PRINT "MITSUBISHI"
 120 BREAK
 130 PRINT "ELECTRIC"
 140 END

```

```

OK
>RUN
MITSUBISHI.....Display by line number 110.
OK
>CONT.....Enter the CONT command.
ELECTRIC.....Display by line number 130.
OK
>

```

EXAMPLE 2

```

OK
>LIST
 100 REM "EXERCISE"
 110 PRINT "MITSUBISHI"
 120 BREAK; PRINT "ELECTRIC"
 130 END

```

→ Not executed.

COLOR

FUNCTION

Specifies the color of character.

FORM

COLOR color code

EXPLANATION

- Specifies the color of character is specified according to color code.
- The list of program is always indicated with white.
- Color codes are as follows:

Color Code	Specified Color	Color Code	Specified Color
0	Black	4	Green
1	Blue	5	Light blue
2	Red	6	Yellow
3	Purple	7	White

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE"
 110 COLOR 4 .....Green is specified.
 120 PRINT "MITSUBISHI" .....MITSUBISHI is displayed.
 130 END
OK
>RUN
MITSUBISHI .....MITSUBISHI is displayed in green characters.
```

EXAMPLE 2

```
OK
>LIST
 100 REM "EXERCISE"
 110 A=0
 120 FOR I=0 TO 6
 130 A=A+1
 140 COLOR A.....Color in variable A is displayed.
 150 X=I*I
 160 PRINT X
 170 NEXT I
 180 END
```

```
OK
>RUN
0.....Displayed blue.
1.....Displayed red.
4.....Displayed purple.
9.....Displayed green.
16.....Displayed light blue.
25.....Displayed yellow.
36.....Displayed white.
```

```
OK
>
```

CLS

TYPE	A
------	---

PRG

FUNCTION

Erases the CRT screen.

FORM

CLS

EXPLANATION

- The CLS command erases the whole display of CRT screen specified by the ZODV command (see P.147).

CLS

TYPE B

PRG

FUNCTION

Erases the CRT screen.

FORM

CLS [1, 2, 3]

EXPLANATION

- The CLS command erases the whole display of CRT screen specified by the ZODV command (see P.147).
- The CLS 1 command erases only presently displayed characters from the screen.
- The CLS 2 command erases only graphic display specified by the VIEW command (see P.208) among the presently shown graphic display.
- When the VIEW command is not provided, the whole graphic display is erased.
- The CLS 3 command erases all of presently displayed characters and graphic display specified by the VIEW command. When the VIEW command is not provided, the whole graphic display is erased.

CLOSE

TYPE | A

PRG

FUNCTION

Closes the channel of K30RSF.

FORM

CLOSE channel number

EXPLANATION

- Among K30RSFs which are connected to specified channels, closes a channel opened by the OPEN command (see P.108).
- Specify [channel number] which is the same as the channel number specified by the OPEN command.
- In [channel number], specify the channel, where K30RSF is connected, among channels 2 ~ 9.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 OPEN 3, $5C, $37, 4 .....K30RSF connected to channel 3 is placed in send /
                                receive enable state.
 120 A=CALL($F, $800C, $6000, $3100) .....Data are sent from K30RSF in channel 3 by SWB
                                command of system subroutine.
 130 CLOSE 3 .....Channel of K30RSF in channel 3 is closed.
 140 END
OK
>
```


CLOSE

TYPE B

PRG

FUNCTION

Closes the K30RSF or RS-232C channel of K31PST.

FORM

CLOSE channel number

EXPLANATION

- Closes a channel opened by the OPEN command (see P.111) among K30RSF or RS-232C ports of K31PST which are connected to specified channels.
- Specify [channel number] which is the same as the channel number specified by the OPEN command.
- For K30RSF, specify [channel number] which is one of channels 2 ~ F. For K31PST, specify [channel number] which is the device number specified by the ZIDV (see P.143) or ZODV (see P.147) command.

CAUTION

1. When a general-purpose terminal is connected to the RS-232C port of K31PST in channel "0" as an I/O console, the CLOSE command cannot be used for that channel.
2. Data, which remain in the send/receive buffer after the CLOSE command is executed, are disabled.

EXAMPLE

```
OK
>LIST
100 OPEN 3, $5C, $37, 4 .....K30RSF connected to channel 3 is placed in send/
110 GOSUB 1000 .....receive enable state.
120 CLOSE 3 .....Channel 3 opened in line number 100 is closed, and
130 STOP .....sending/receiving by K30RSF is disabled.
1000 REM "EXERCISE"
1010 RETURN
OK
>
```

CLOSE

TYPE	C
------	---

PRG

FUNCTION

Closes the channel of RS-232C or RS-422.

FORM

CLOSE channel number

EXPLANATION

- Closes the channel of RS-232C or RS-422 opened by the OPEN command (see P.114).
- Specify [channel number] which is the same as the channel number specified by the OPEN command.
- In [channel number], specify channel 1 or 2 for RS-232C and channel 3 for RS-422.

CAUTION

1. For channel 0 of RS-232C, the CLOSE command cannot be used.
2. Data, which remain in the send/receive buffer after the CLOSE command is executed, are disabled.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 OPEN $2, $5C, $37, 4 .....Channel 2 of RS-232C is opened and placed in send/
120 FOR I=0 TO 10 .....receive enable state.
130 A=I*I
140 ZODV2
150 PRINT A
160 NEXT I
170 ZTIME 300
180 CLOSE $2 .....Channel 2 of RS-232C is closed.
190 END
OK
>
```

END

PRG

FUNCTION

Ends the run of program and returns control to OS side.

FORM

END

EXPLANATION

- By executing the END command, the run of program is completed and the command accept state is restored.
- When the END command is executed during the run of multi task, control is returned to OS side and the task is not run until the start condition selected during "setting of multitask" is generated again.

FOR...NEXT

FUNCTION

Executes a series of instructions consecutively by the number of specified times.

FORM

```
FOR variable name=n1 TO n2[STEP n3]
```

```
⋮
```

```
NEXT variable name
```

n1, n2 and n3 are numerical expressions.

EXPLANATION

- (variable name) is used as a counter. The first numerical expression n1 indicates the initial value of counter, and the second numerical expression n2 indicates the last value of counter.
- When the statement located between the FOR statement and NEXT statement is executed, an increment n3 specified in STEP is added to the value of counter, and the resultant counter value is compared with the last value n2.
- When the counter value is less than or equal to the last value, the same operation is repeated beginning with the statement following the FOR statement. When the counter value is greater than the last value, the statement following the NEXT statement is executed.
- When STEP is omitted, the increment of counter is regarded as 1.
When the value specified in STEP is negative, the last value should be less than the initial value. In this case, the value of counter is reduced per loop and the loop continues until the counter value becomes less than the last value.
- When the value in STEP is positive and the initial value is greater than the last value, or when the value in STEP is negative and the initial value is less than the last value, the statement located between FOR statement and NEXT statement is executed only once.
- The FOR...NEXT loop can be nested. FOR...NEXT can be placed inside another FOR...NEXT loop. When the loop is nested, variable name used for each loop should be different. One FOR...NEXT statement should be located inside another FOR...NEXT statement.
- The appearance of NEXT statement, which has no corresponding FOR statement, results in error and stop of program run.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 FOR A=1 TO 10 STEP 2
```

1 is first assigned to variable A, and then with the assigned value increasing by two, values up to 10 are assigned in due order. Each time a value is assigned to variable A, line is executed up to line number 130, and then execution returns to line number 110. When the value in variable A exceeds 10, execution jumps to line number 140 and the program ends.

```
120 PRINT A
130 NEXT A
140 END
```

```
OK
>RUN
```

- 1
- 3
- 5
- 7
- 9

```
OK
>
```

GOTO

FUNCTION

Transfers the order of program run unconditionally to the specified line number.

FORM

GOTO line number

EXPLANATION

- The order of program run transfers unconditionally to the line specified in [line number] and the program is run beginning with a statement in the specified line.
- When the statement in the line specified in [line number] is a non-executable statement (REM command (see P.124), the program is run beginning with the next executable statement.
- In the case of ONGOTO command (see P.107), the value of line number is represented by an expression.

EXAMPLE

```

OK
>LIST
100 REM "EXERCISE"
110 X=1
120 PRINT X
130 X=2*X
140 GOTO 120 ..... Execution is transferred to line number 120.

OK
>RUN
  1
  2
  4
  8
OK ..... By pressing the [CONT] / [C] keys, execution
> ..... returns to command accept state.

```

GOSUB....RETURN

FUNCTION

Departs execution to subroutine and returns it to its departure point.

FORM

```
GOSUB   line number
      ⋮
RETURN
```

EXPLANATION

- [line number] indicates the line number at the head of subroutine.
- The same subroutine can be called any desired times in a program.
- One subroutine can call another subroutine.
- Subroutines can be nested in any desired depth to the limit of memory capacity.
- The RETURN statement returns execution to the statement following the GOSUB statement.
- In the case of ONGOSUB command (see P.106), the value of line number is represented by an expression.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 A=1
120 B=2
130 C=3
140 D=4
150 GOSUB 300 .....Execution is transferred to line number 300.
160 A=8
170 B=12
180 C=9
190 D=15
200 GOSUB 300 .....Execution is transferred to line number 300.
210 END
300 X=A*B+C/D
310 PRINT "A*B+C/D="X
320 RETURN .....When subroutine starting at line number 300 is called
OK                                     by line number 150, execution transfers to line number
>                                     160. When it is called by line number 200, execution
                                     returns to line number 210.
```

IF

FUNCTION

Selects a statement, which runs a program, according to the result of expression.

FORM

IF[.] expression 1 comparing operator expression 2 statement

EXPLANATION

- When the condition of comparing operator between expression 1 and expression 2 holds, the statement located behind expression 2 is executed. When the condition does not hold, the program is run beginning with the statement in the next line number.
- The expression 1 and expression 2 are numerical expressions, and the forms (variable forms) of expression 1 and expression 2 should be the same.
- When combining the comparing operator and the logical operator, parenthesize the logical operator.
- When handling a real expression in the IF statement, provide a period (.) behind IF.

EXAMPLE 1

```

OK
>LIST
100 REM "EXERCISE 1"
110 A=1
120 FOR B=1 TO 9
130 C=A*B
140 PRINT #3, C,
150 NEXT B
160 A=A+1
170 IF A=10 END.....When value in variable A is 10, program ends. When
180 PRINT it is not 10, line number 180 is executed.
190 GOTO 120

```

```

OK
>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
>

```


EXAMPLE 2

Comparing operator and logical operator are combined

```
OK
>LIST
100 REM "EXERCISE 2"
110 INPUT A, B, C, D
120 IF (A!B) # (C&D) PRINT "UNEQUAL";GOTO110
130 PRINT "EQUAL"
140 END
```

When the value of logical sum of variable A and variable B and the value of logical product of variable C and variable D are not equal, "UNEQUAL" is displayed and execution returns to line number 110.

```
OK
>RUN
4
2
-1
8
UNEQUAL
8
2
10
15
EQUAL
OK
>
```

EXAMPLE 3

Real number is handled

```
OK
>LIST
100 REM "EXERCISE 3"
110 A=$C000
120 B=$E000
130 INPUT A(0), B(0)
140 A(1)=ASIN (A(0))
150 B(1)=ACOS (B(0))
160 IF A (1) =B (1) GOTO190
170 PRINT "UNEQUAL"
180 END
190 PRINT "EQUAL"
200 END
```

When the contents of addresses C004 to C007 are equal to the contents of addresses E004 to E007, execution transfer to line number 190.

```
OK
>RUN
0.866
0.5
UNEQUAL

OK
>RUN
0.841471
0.540302
EQUAL
OK
>
```

INPUT

TYPE A

PRG

FUNCTION

Enables data entry through the keyboard during run of program.

FORM

`INPUT ["prompt character string",] variable name 1[,["prompt character string",] variable name 2]`

EXPLANATION

- The INPUT command temporarily stops the run of program and waits for data entry through the keyboard.
- Specify a integer variable in [variable name].
- Entered integer constant data is -32767 to 32767.
- An expression, which connects integer constant and basic variable by operators for four operations (addition, subtraction, multiplication and division), can also be used as entered data.
- Data entry for one variable is completed by pressing the **CR** key. The entered data is assigned to the variable specified in the variable name.
- When two or more INPUT commands are present consecutively, the INPUT command performs line feed and waits for the next data entry each time an entry for one variable is completed.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 INPUT A
120 INPUT B
130 INPUT C
140 PRINT A, B, C
150 END
```

.....Entered data are assigned to variables A, B and C.
This is the same as INPUT "A=A", "B=B", "C=C".

```
OK
>RUN
A: 10
B: 150
C: A+B
```

.....When keys are pressed in the following order ; 10 **CR**
150 **CR** A+B **CR**.

```
10 150 160
```

.....Display by line number 140.

INPUT

TYPE B

PRG

FUNCTION

Enables data entry through the keyboard during run of program.

FORM

INPUT ["prompt character string",] variable name 1[,["prompt character string",] variable name 2]

EXPLANATION

- The INPUT command temporarily stops the run of program and waits for data entry through the keyboard.
- Integer variable, real number variable and character string variable can be used in [variable name].
- Entered integer constant data is -32767 to 32767.
- An expression, which connects integer constant and basic variable by operators for four operations (addition, subtraction, multiplication and division), can also be used as entered data.
- Data entry for one variable is completed by pressing the **CR** key. The entered data is assigned to the variable specified in the variable name.
- When two or more INPUT commands are present consecutively, the INPUT command performs line feed and waits for the next data entry each time an entry for one variable is completed.

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE 1"
110 INPUT "A= ", A
120 INPUT "B= ", B
130 INPUT "C= ", C
140 PRINT A, B, C
150 END
```

.....Entered data are assigned to variables A, B and C. This is the same as INPUT "A= ", A, "B= ", B, "C= ", C.


```
OK
>RUN
A=10
B=150
C=A+B
```

.....When keys are pressed in the following order ;10 **CR** 150 **CR** A+B **CR**

```
10 150 160.....Display by line number 140.
```

EXAMPLE 2

```
OK
>LIST
100 REM "EXERCISE 2"
110 A=SC000
120 B=$D000
130 C=$E000
140 INPUT A (0) .....Real number is assigned to addresses C000 to C003.
150 INPUT CS .....Character string is assigned to E000 and following
                    addresses.
160 B (0) =SQRT (A (0))
170 PRINT CS, . B (0)
180 END
```

```
OK
>RUN
12.34 .....Data entry by line number 140.
SQUARE ROOT= .....Data entry by line number 150.
SQUARE ROOT=0.3512831E 01 .....Display by line number 170.
```

Assuming that the variable of line number 140 is A, when a real number (140 INPUT A) is entered, the fraction part is truncated and the remaining part is assigned to the variable A.

LCOPY

TYPE	A
------	---

PRG

FUNCTION

Prints the data displayed on the screen through the printer.

FORM

LCOPY

EXPLANATION

- Prints the data presently displayed on the screen through the printer.
- In the case of color display, the data are not printed in different colors.

CAUTION

Although it takes several minutes to execute the LCOPY command, do not change the screen during the execution of this command.

LCOPY

TYPE B

PRG

FUNCTION

Prints the data displayed on the screen through the printer.

FORM

LCOPY

EXPLANATION

- The data presently displayed on the screen is printed through the printer.
- Usable printer is only K6PR or K7PRE which is connected to the RS-232C port or parallel interface port of K31PST. When a general-purpose printer is connected, the LCOPY command cannot be used.
- In the case of color display, the data are not printed in different colors.
- Before executing the LCOPY command, it is required to specify the channel number of K31PST and the channel of RS-232C by the ZODV command (see P.147).

CAUTION

Although it takes several minutes to execute the LCOPY command, do not change the screen during the execution of this command.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 ZODV $52.....For setting, see ZODV.
 120 LCOPY
 130 END
OK
>
```

LET

FUNCTION

Assigns the value of an expression to a variable.

FORM

[LET] variable=expression

EXPLANATION

- The value calculated by the expression is assigned to the variable specified in the variable name.
- LET can be omitted.

EXAMPLE

```

OK
>LIST
 100 REM "EXERCISE"
*110 LET A=0.....0 is assigned to variable A.
*120 LET X=A*A.....Value calculated by A*A is assigned to variable X.
 130 PRINT X
*140 LET A=A+1.....1 is added to the content of variable A.
 150 GOTO 100
 160 END

OK
>RUN
 0
 1
 4
 9
16
25
.....
OK
>

```

The line with * can be used without specifying LET.

```

110 A=0
120 X=A*A
140 A=A+1

```

LOCATE

TYPE A

PRG

FUNCTION

Specifies the cursor position on the screen.

FORM

LOCATE line position, column position

EXPLANATION

- The line position and column position can also be specified by expressions.
- Ranges of the line position and column position are as shown in the table below:

Mode	Standard	Double	Quadruple	GPP Character
Line	0 to 19	0 to 9	0 to 4	0 to 23
Column	0 to 35	0 to 17	0 to 8	0 to 79

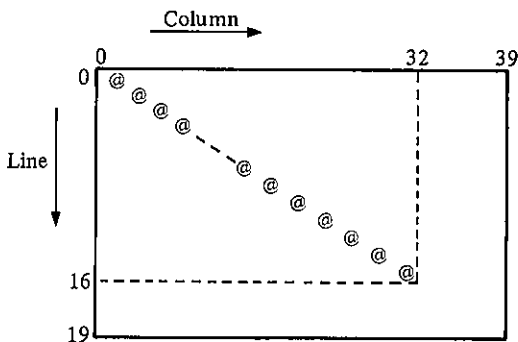
- If a value outside the usable range is specified, the cursor position does not change.

EXAMPLE

The standard 640 x 400 mode screen has been specified.

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 FOR I=0 TO 16
 130 LOCATE I, I*2
 140 PRINT "@"
 150 NEXT I
 160 END
OK
>RUN
```

As shown at right, @ is displayed from 0, 0 to 16, 32.



LOCATE

TYPE B

PRG

FUNCTION

Specifies the cursor position on the screen.

FORM

LOCATE line position, column position

EXPLANATION

- The line position and column position can also be specified by expressions.
- Ranges of the line position and column position are as shown in the table below:

CRT Mode		Screen Mode by ZDSP Command		
		Standard	Double	Quadruple
640 x 400 mode	Line	0 to 19	0 to 9	0 to 4
	Column	0 to 39	0 to 19	0 to 9
1024 x 400 mode	Line	0 to 19	0 to 9	0 to 4
	Column	0 to 63	0 to 31	0 to 15
General-purpose CRT	Line	0 to 24		
	Column	0 to 79		

For ZDSP command, (see P.155).

- If a value outside the usable range is specified, the cursor position does not change.

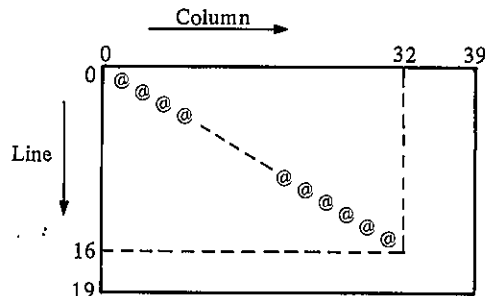
EXAMPLE 1

Standard 640 x 400 mode screen has been specified.

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 CLS
 120 FOR I=0 TO 16
 130 LOCATE I, I*2
 140 PRINT "@"
 150 NEXT I
 160 END
```

```
OK
>RUN
```

As shown at right, @ is displayed from 0, 0 to 16, 32.



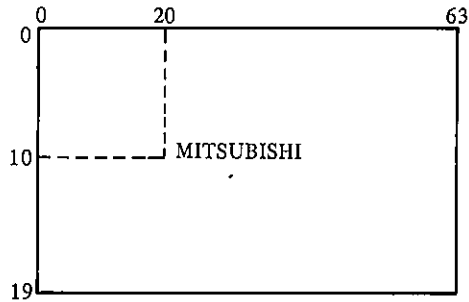
EXAMPLE 2

Standard 1024 x 400 mode screen has been specified

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 A=$E000
 120 INPUT AS
 130 CLS
 140 LOCATE 10, 20
 150 PRINT AS
 160 END
```

```
OK
>RUN
MITSUBISHI .....Data entry by line number 120.
```

Display is as shown in the figure below.



LOCATE

TYPE C

PRG

FUNCTION

Specifies the cursor position on the screen.

FORM

LOCATE line position, column position

EXPLANATION

- The line position and column position can also be specified by expressions.
- Ranges of the line position and column position are as shown in the table below:

	Screen Mode
Line	0 to 24
Column	0 to 79

- If a value outside the usable range is specified, the cursor position does not change.

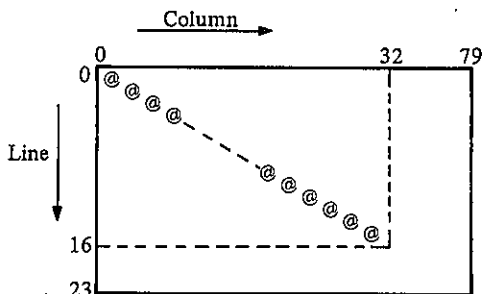
EXAMPLE

Standard 640 x 400 mode screen has been specified.

```
OK
>LIST REM "EXERCISE"
 110 CLS
 120 FOR I=0 TO 16
 130 LOCATE 1, 1*2
 140 PRINT "@"
 150 NEXT I
 160 END
```

```
OK
>RUN
```

As shown at right, @ is displayed from 0, 0 to 16, 32.



LPRINT

TYBP	A
------	---

PRG

FUNCTION

Prints output data through the printer.

FORM

LPRINT [expression[, expression].....] [;]

EXPLANATION

- The LPRINT command is the same as the PRINT command (see P.106) except that output data is printed through the printer.
- When the GPP mode is selected by the ZDSP command and the LPRINT command is executed in this mode, only characters, of which key codes are one byte, are printed.
- For the LPRINT command, the LOCATE command (see P.99) cannot be used.

LPRINT

TYPE	B
------	---

PRG

FUNCTION

Prints output data through the printer.

FORM

```
LPRINT [expression[, expression] . . . . .] [;]
```

EXPLANATION

- The LPRINT command is the same as the PRINT command (see P.121) except that output data is printed through the printer.
- For the LPRINT command, the LOCATE command (see P.100) cannot be used.

LPRINT

TYPE	C
------	---

PRG

FUNCTION

Prints output data through the printer.

FORM

LPRINT [expression[, expression]] [;]

EXPLANATION

- The LPRINT command is the same as the PRINT command (see P.121) except that output data is printed through the printer.
- For the LPRINT command, the LOCATE command (see P.102) cannot be used.

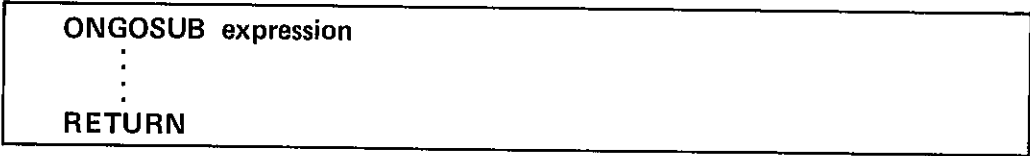
ONGOSUB....RETURN

PRG

FUNCTION

Departs execution to subroutine and returns it to its departure point.

FORM



EXPLANATION

- Departs execution to a subroutine which begins with the line number indicated by the value of expression.
- The RETURN statement returns to the statement located in the line following the ONGOSUB statement.
- The same subroutine can be called any desired times in the program.
- A subroutine can call another subroutine.
- Subroutines can be nested in any desired depth to the limit of memory capacity.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 C=180
120 ONGOSUB C+20.....Execution is transferred to line number of C+20
130 C=C+20.....value.
140 IF C=240 GOTO 160
150 GOTO120
160 END
200 PRINT "PROCESS A"
210 RETURN
220 PRINT "PROCESS B"
230 RETURN
240 PRINT "PROCESS C"
250 RETURN

OK
>RUN
PROCESS A
PROCESS B
PROCESS C
OK
>
```

ON GOTO

PRG

FUNCTION

Transfers the order of program run unconditionally to the line number specified by an expression.

FORM

ONGOTO expression

EXPLANATION

- Transfers execution unconditionally to the line number specified by the calculated value of expression.
- When the statement in the line specified by the calculated value of expression is a non-executable statement (REM command (see P.124)), the program is run beginning with the next executable statement.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 INPUT A
 120 ONGOTO A%2 *20+130 .....Since the value of A%2 is 0 or 1, 130 or 150 is specified
 130 PRINT "EVEN NUMBER" .....as line number.
 140 GOTO 110
 150 PRINT "ODD NUMBER"
 160 END.....When "ODD NUMBER" is displayed, execution is
                           completed.
OK
>RUN
A: 200
  EVEN NUMBER
A: 50
  EVEN NUMBER
A: 333
  ODD NUMBER
OK
>
```


OPEN

TYPE	A
------	---

PRG

FUNCTION

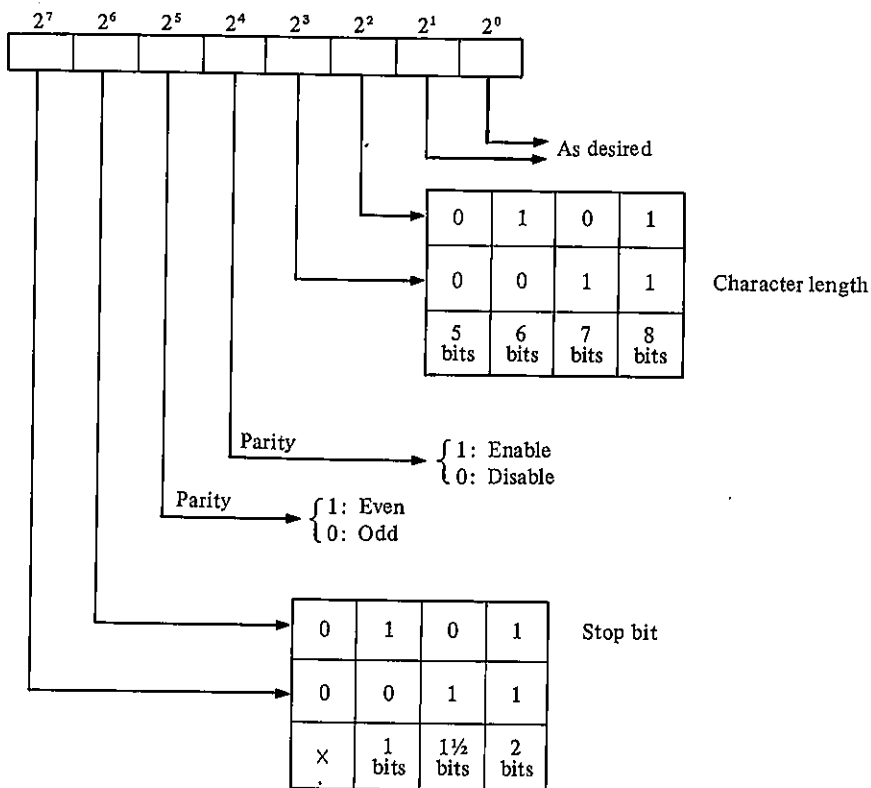
Opens the channel of K30RSF and enables sending/receiving operations.

FORM

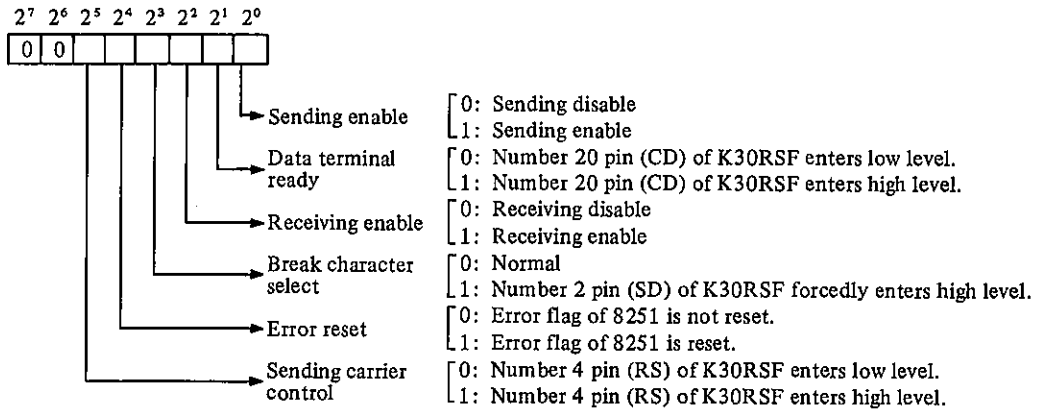
OPEN channel number, variable 1, variable 2, variable 3

EXPLANATION

- Opens the channel of K30RSF specified in [channel number].
- When the block data receiving (SRB (see P. 285)) and block data sending (SWB (see P. 289)) of system subroutine are used, it is required to open the channel by executing the OPEN statement.
- [channel number] is one of channels 2 to 9.
- Variable 1 is for the setting of communication mode, and configured as shown in the figure below when converted into binary.



- Variable 2 is for the setting of command, each digit is 1, and has the following meaning when converted into binary.



- Variable 3 is for the setting of baud rate and as shown below.

↙	1	2	3	4	5	6
Baud rate	300	600	1200	2400	4800	9600

NOTE

1. For details of communication mode and command, see the instruction manual provided for LSI8251.
2. Normally, commands seem to be as indicated below:
 - 36H – only when receiving is performed
 - 33H – only when sending is performed
 - 37H – both sending and receiving are performed

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE"
110 OPEN 3, $5C, $37, 4.....Predetermined setting is sent to K30RSF in channel 3
                               to enable sending.
120 A=CALL ($F, $80C, $6000, $3100).....System subroutine SWB is called.
130 END
OK
>
```

Data beginning with the address \$6000 of channel 0 are sent to the equipment, which is connected to the K30RSF in channel 3, by use of the block data sending (SWB) command. After opening the channel, make communication by use of the SWB and SRB commands.

EXAMPLE 2

When it is desired to open the K6PRE or K7PRE in the BASIC program, set as indicated below.

```
OPEN X, $7F, $37, 4.....K6PRE
OPEN X, $CF, $37, 4.....K7PRE
```

(X changes depending on the connected channel.)

OPEN

TYPE B

PRG

FUNCTION

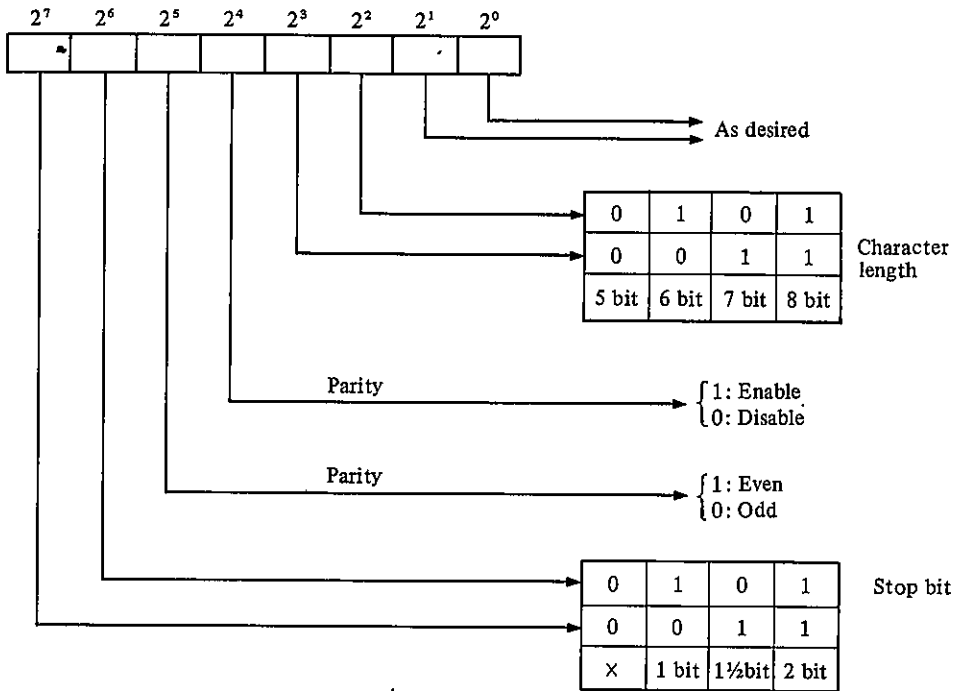
Opens the channel of K30RSF or the RS-232C channel of K31PST and enables sending/receiving operations by setting the communication mode.

FORM

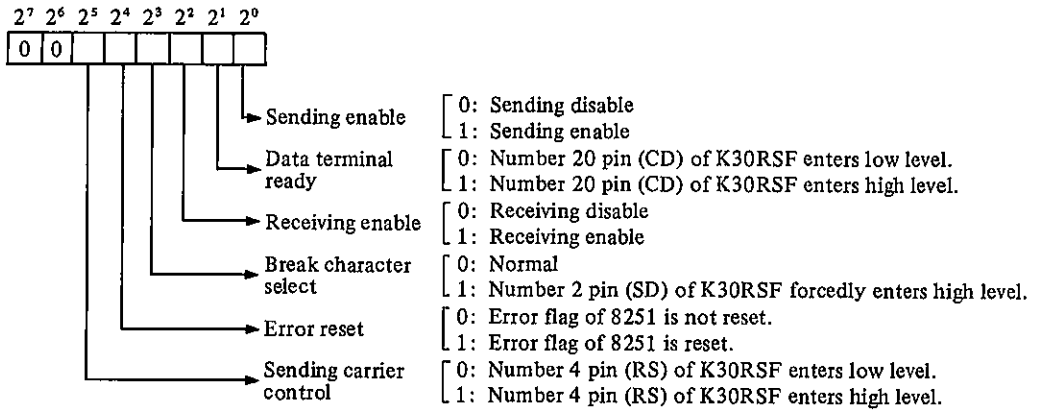
OPEN channel number, variable 1, variable 2, variable 3

EXPLANATION

- Opens the channel of K30RSF or the RS-232C channel of K31PST specified in [channel number].
- When the block data receiving (SRB (see P.287)) and block data sending (SWB (see P.291)) of system subroutine are used, it is required to open the channel by executing the OPEN statement.
- [channel number] is one of channels 2 to F for K30RST, and a device number specified by the ZIDV command (see P.143) or ZODV command (see P.147) for RS232C of K31PST.
- Variable 1 is for the setting of communication mode and configured as shown in the figure below when converted into binary.



- Variable 2 is for the setting of command, each digit is 1, and has the following meaning when converted into binary.



- Variable 3 is for the setting of baud rate and as shown below.

	1	2	3	4	5	6
Baud rate	300	600	1200	2400	4800	9600

NOTE

1. For details of communication mode and command, see the instruction manual provided for LSI8251.
2. Normally, commands seem to be as indicated below:
 36H – only when receiving is performed
 33H – only when sending is performed
 37H – both sending and receiving are performed
3. When a general-purpose terminal is used as I/O console for the RS-232C port of K31PST in channel 0, the OPEN command cannot be used for that channel.
4. Data, which remains in the sending/receiving buffer upon execution of OPEN command, is disabled.

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE"
110 OPEN 3, $5C, $37, 4 .....Predetermined setting is sent to K30RSF in channel 3
120 A=$E300 .....to enable sending.
130 A(0)=$6000
140 A(1)=$3100 .....3 channel data length is 256 bytes.
150 A(2)=10
160 B=CALL($0, $900C, 4, $E300) .....System subroutine SWB is called.
170 END
OK
>RUN
OK
```

Data beginning with the address \$6000 of channel 0 are sent to the equipment, which is connected to the K30RSF in channel 3, by use of the block data sending (SWB) command. After opening the channel, make communication by use of the SWB and SRB commands.

EXAMPLE 2

When it is desired to open the K6PRE or K7PRE in BASIC, set as indicated below.

```
OPEN X, $7F, $37, 4 ..... K6PRE
OPEN X, $CF, $37, 4 ..... K7PRE
```

(X changes depending on a connected channel.)

OPEN

TYPE C

PRG

FUNCTION

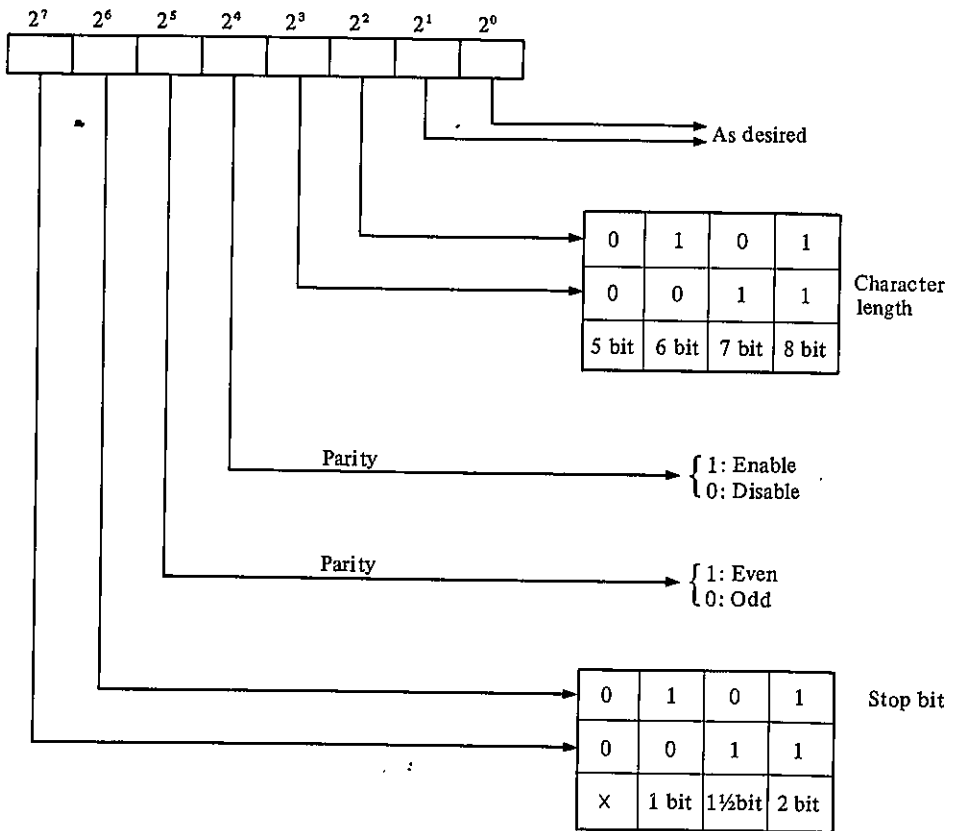
Opens the RS-232C or RS-422 channel and enables sending/receiving operations.

FORM

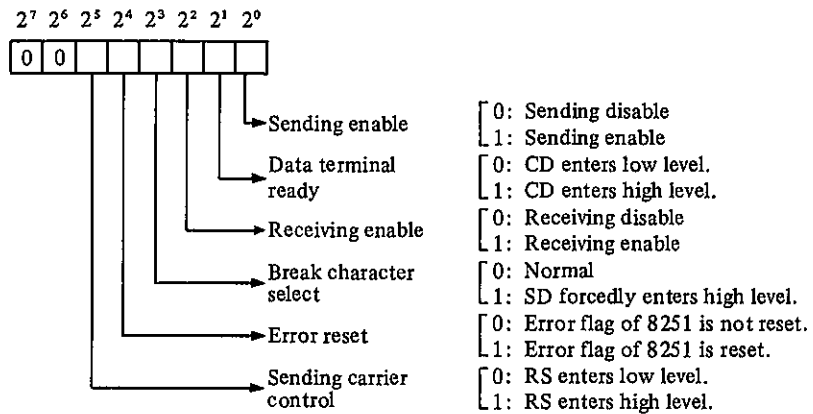
OPEN channel number, variable 1, variable 2, variable 3

EXPLANATION

- Opens the RS-232C or RS-422 channel specified in [channel number].
- When the block data receiving (SRB (see P.287)) and block data sending (SWB (see P.291)) of system subroutine are used, it is required to open the channel by executing the OPEN statement.
- [channel number] is one of channels 1 to 3.
- Variable 1 is for the setting of communication mode and configured as shown in the figure below when converted into binary.



- Variable 2 is for the setting of command, each digit is 1, and has the following meaning when converted into binary.



- Variable 3 is for the setting of baud rate and as shown below.

	1	2	3	4	5	6
Baud rate	300	600	1200	2400	4800	9600

NOTE

1. For details of communication mode and command, see the instruction manual provided for LSI8251.
2. Normally, commands seem to be as indicated below:
 - 36H — only when receiving is performed
 - 33H — only when sending is performed
 - 37H — both sending and receiving are performed
3. When a general-purpose terminal is used as I/O console for the RS-232C port of K31 PST in channel 0, the OPEN command cannot be used for that channel.
4. Data, which remains in the sending/receiving buffer upon execution of OPEN command, is disabled.

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE"
110 OPEN 2, $5C, $37, 4.....Predetermined setting is sent to RS-232C in channel 2
120 A=$E300                    to enable sending.
130 A(0)=$6000
140 A(1)=$2100.....2 channel data length is 256 bytes.
150 A(2)=10
160 B=CALL ($0, $800C, 4, $E300).....System subroutine SWB is called.
170 END
OK
>RUN
OK
```

Data beginning with the address of \$6000 of channel 0 are sent to the equipment, which is connected to the channel 2 of RS-232C, by use of the block data sending (SWB) command. After opening the channel, make communication by the SWB and SRB commands.

EXAMPLE 2

When it is desired to open the K6PRE or K7PRE in BASIC, set as indicated below.

```
OPEN X, $7F, $37, 4.....K6PRE
OPEN X, $CF, $37, 4.....K7PRE
```

(X changes depending on a connected channel.)

POKE

FUNCTION

Writes one-byte data to the specified address of memory.

FORM

POKE expression 1, expression 2

EXPLANATION

- One-byte data in [expression 2] is written to the memory in the address indicated in [expression 1] in the same channel as the program.
- [expression 1] and [expression 2] are integer expressions.
- The range of memory addresses in [expression 1] is 0 to 65535 (\$0 to \$FFFF) addresses. However, since 32768 (address \$8000) or greater number is negative, provide "\$" mark for such a number and specify it in hexadecimal notation.
- Numeric values, which can be specified in [expression 2] are 0 to 255.
- As a command which has a function reverse to the POKE command, there is a PEEK command (see P.129).

WARNING
Among memory addresses specified in [expression 1], addresses \$0 to \$5FFF and addresses \$6800 to \$7FFF are used on the OS side. Therefore, never write data to such addresses.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 A=$E000
 120 INPUT X
 130 POKE A, X.....One-byte data, which is entered in line number 120, is
 140 END                          stored in address E000.
OK
>RUN
X: 1
OK
>
```

PRINT

TYPE | A

PRG

FUNCTION

Displays output data on the CRT screen.

FORM

```
PRINT [list of expressions] [,]  
?[list of expressions] [,]
```

EXPLANATION

- Data are output on the CRT screen specified by the ZODV command.
- When [list of expressions] is omitted, only line feed is performed.
- Expressions in [list of expressions] are numeric value, variable or character string expressions. Also, these expressions can be used together.
- When using a character string expression, be sure to put it in double quotes (""). ("character string expression")
- When a real number is displayed, it is displayed in exponent to seven significant digits.
- The PRINT command normally performs line feed. However, line feed can be inhibited by providing a comma (,).
- Display forms can be specified by the PRINT command as shown below:

Specifying Method	Content of Display
\$ expression (variable)	Value of expression (variable) is displayed in hexadecimal four digits.
? expression (variable)	Value of expression (variable) is displayed in hexadecimal two digits.
. expression (variable)	Value of expression (variable) is displayed in real number form.
# expression (variable)	Value of expression (variable) is displayed in its printing width. (Initial value is 6.)
* expression (variable)	Value of expression (variable) is regarded as key code and displayed in characters which correspond to key codes.

- When the form is not specified, value of expression is displayed in decimal six digits.
- In the case of \$ expression, ? expression and * expression, value of expression is displayed without providing a space on the left.
- # expression is effective only for a normal decimal integer. The printing width includes a positive or negative sign. However, when the number is positive, the sign part is displayed as a space.
- In regards to # expression, when the number of digits of output data is greater than a specified printing width, the designation of printing width is ignored and all digits are displayed.

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 A=10
 120 B=5
 130 PRINT A, B.....Contents of variable A and variable B are displayed
 140 PRINT A+B, A*B.....without line feed.
 150 END

OK
>RUN
   10          5.....Display by line number 130.
   15          50.....Display by line number 140.

OK
>
```

EXAMPLE 2

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 A=INKEY.....Entered data is assigned to variable A.
 120 PRINT *A, .....Content of variable A is regarded as key code, and
 130 GO TO 110.....character corresponding to the code is displayed.

OK
>RUN
NEW YEAR'S DAY OF 1984.....Display when "NEW YEAR'S"DAY OF 1984" is
OK.....entered through keyboard.
>
```

EXAMPLE 3

```
OK
>LIST
 100 REM "EXERCISE 3"
 110 A=S1234
 120 PRINT A.....Content of variable A is displayed in decimal six
 130 PRINT $A.....digits.
 140 PRINT ?A.....Content of variable A is displayed in decimal four
 150 END.....digits.

OK
>RUN
 4660.....Display by line number 120.
1234.....Display by line number 130.
34.....Display by line number 140.

OK
>
```

EXAMPLE 4

```
OK
>LIST
100 REM "EXERCISE 4-1"
110 FOR I=0 TO 5
120 PRINT #4, I .....Content of count value I is displayed in four digits.
140 NEXT I
150 END
```

Line feed is inhibited.

```
OK
>RUN
0 1 2 3 4 5 .....Display by line number 120.
OK
>
```

```
OK
>LIST
100 REM "EXERCISE 4-2"
110 A=1234
120 PRINT #3, A .....Content of variable A is displayed in three digits.
130 END
```

```
OK
>RUN
1 2 3 4 .....Since content of variable A cannot be displayed in
three digits, designation of three digits is ignored, and
all digits are displayed.
OK
>
For sign, a space is provided at the head.
```

PRINT

TYPE	B
------	---

PRG

FUNCTION

Displays output data on the CRT screen.

FORM

```
PRINT [list of expressions] [,]  
PRINT .[(expression 1[, expression 2] )] real variable[,]  
?[list of expressions] [,]  
?.[(expression 1[, expression 2] )] real variable[,]
```

EXPLANATION

- Data are output to the CRT screen specified by the ZODV command.
- When [list of expressions] is omitted, only line feed is performed.
- Expressions in [list of expressions] are numeric value, variable or character string expressions. Also, these expressions can be used together.
- When using a character string expression, be sure to put it in double quotation (""). ("character string expression")
- A real number is displayed in exponent to seven significant digits when [expression 1, expression 2] is omitted.
- A real number is displayed in seven digits of integer part and decimal part when [expression 1, expression 2] is specified.
- Specify the number of digits of integer part in [expression 1] and the number of digits of decimal part in [expression 2].
- When [expression 2] is omitted, only the integer part is displayed.
- The PRINT command normally performs line feed. However, line feed can be inhibited by providing a comma (,).
- Display forms can be specified by the PRINT command as shown below:

Specifying Method	Content of Display
\$ expression (variable)	Value of expression (variable) is displayed in hexadecimal four digits.
? expression (variable)	Value of expression (variable) is displayed in hexadecimal two digits.
. expression (variable)	Value of expression (variable) is displayed in real number form.
# expression (variable)	Value of expression (variable) is displayed in its printing width. (Initial value is 6.)
* expression (variable)	Value of expression (variable) is regarded as key code and displayed in characters which correspond to key code.

- When the form is not specified, value of expression is displayed in decimal six digits.
- In the case of \$ expression, ? expression and * expression, value of expression is displayed without providing a space on the left.
- # expression is effective only for a normal decimal integer. The printing width includes a positive or negative sign. However, when the number is positive, the sign part is displayed as a space.
- When the number of digits of output data is greater than a specified printing width, the designation of printing width is ignored and all digits are displayed.

EXAMPLE 1

```

OK
>LIST
 100 REM "EXERCISE 1"
 110 A=10
 120 B=5
 130 PRINT A, B .....Contents of variable A and variable B are displayed
                               without line feed.
 140 PRINT A+B, A*B
 150 END
OK
>RUN
 10          5 .....Display by line number 130.
 15        50 .....Display by line number 140.
OK
>

```

EXAMPLE 2

```

OK
>LIST
 100 REM "EXERCISE 2"
 110 A=SC000
 120 A [0] =12.34
 130 PRINT. A [0] .....When expression 1 and expression 2 are omitted, data
                               is displayed in exponent form.
 140 PRINT. (2) A [0] .....When expression 2 is omitted, only integer part is
                               displayed.
 150 PRINT. (2, 2) A [0] .....Data is displayed in the number of digits specified
                               in expression 1 and expression 2.
 160 END
OK
>
>RUN
 0.1233999E 02 .....Display by line number 130.
 12 .....Display by line number 140.
 12.34 .....Display by line number 150.
OK
>

```

EXAMPLE 3

```
OK
>LIST
100 REM "EXERCISE 3"
110 A=INKEY
120 PRINT *A, .....Content of variable A is regarded as key code, and
                        character corresponding to the code is displayed.

130 GOTO 110
OK
>RUN
NEW YEAR'S DAY OF 1985 .....Display when "NEW YEAR'S DAY OF 1985" is
OK                                     entered through keyboard
>
```

EXAMPLE 4

```
OK
>LIST
100 REM "EXERCISE 4"
110 A=$1234
120 PRINT A .....Content of variable A is displayed in decimal six
                        digits.
130 PRINT $A .....Content of variable A is displayed in decimal four
                        digits.
140 PRINT ?A .....Content of variable A is displayed in decimal two
                        digits.
150 END
OK
>RUN
4660 .....Display by line number 120.
1234 .....Display by line number 130.
34 .....Display by line number 140.
OK
>
```

EXAMPLE 5

```
OK
>LIST
100 REM "EXERCISE 5-1"
110 FOR I=0 TO 5
120 PRINT #2, I, .....Content of count value I is displayed in four digits.
140 NEXT I
150 END
      |
      |-----> Line feed is inhibited.
OK
>RUN
0 1 2 3 4 5 .....Display by line number 120.
OK
>
```

```
OK
>LIST
100 REM "EXERCISE 5-2"
110 A=1234
120 PRINT #3, A .....Content of variable A is displayed in three digits.
130 END
OK
>RUN
1234 .....Since content of variable A cannot be displayed in
OK                                     three digits, designation of three digits is ignored, and
>                                     all digits are displayed.
                                         For sign a space is provided at the head.
```


REM

PRG

FUNCTION

Gives a comment in the program.

FORM

```
REM [""] comment ["]
```

EXPLANATION

- The REM command is used to write a comment so that the user can easily look at the program.
- The REM command may also be written as a mark at the jump destination of GOTO (see P.89) or GOSUB (see P.90). In this case, execution is started at the first executable statement located behind the REM command.
- Since a semicolon (;) is a part of comment in the REM command, the semicolon cannot be used to divide the REM command in order to continue another statement.
- Be sure to enclose the comment with double quotes (""). ("Comment")
- Although the REM command does not affect the processing of program, the process time may be elongated approximately 0.5 to 2 ms (depending on the number of characters in a comment).

EXAMPLE

```
OK
>LIST
 100 REM REM KOMANDO NO SETUMEI
 110 REM "EXPLANATION OF REM
      COMMAND" .....When characters other than ASCII code are used,
 120 PRINT "MITSUBISHI"          enclose characters in double quotes.
 130 END

OK
>RUN
MITSUBISHI
OK
>
```

STOP

FUNCTION

Stops run of program during debugging of program and restores the command accept state.

FORM

STOP

EXPLANATION

- The STOP command is used for debugging, etc. and may be used anywhere in the program.
- The STOP command cannot be used during run of multi task.
- If the CONT command (see P.59) is used after the command accept state is restored, the run of program is not resumed.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 PRINT "MITSUBISHI"
 120 STOP.....Program run is stopped.
 130 PRINT "ELECTRIC"
 140 END

OK
>RUN.....Program is run beginning with line number 100.
MITSUBISHI.....Program is displayed on the screen by number 110.
OK
>RUN
MITSUBISHI
OK
>RUN 130.....Program is run beginning with line number 130.
ELECTRIC
OK
>
```

3.2.2 Program commands which are functions

- The formats of program commands, which are functions, are as shown below.

Format of Assignment Statement

```
100 A=INKEY
```

Format of IF Statement

```
100 IF $31=INKEY GOTO **
```

- The program commands, which are functions, are as follows:

Command Name	Reference Page
CALL	127
INKEY	128
PEEK	129
SIZE	130

CALL

PRG

FUNCTION

Transfers control of execution to a machine language subroutine which is loaded on the memory.

FORM

CALL (variable 1, variable 2, variable 3[, variable 4])

EXPLANATION

- In [variable 1], specify the channel of memory where the machine language subroutine to be called exists.
- In [variable 2], specify the head address of memory where the machine language subroutine to be called exists.
- In [variable 3], specify the value which is used for each machine language subroutine and set to (D) and (E) registers.
- In [variable 4], specify the value which is used for each machine language subroutine and set to (B) and (C) registers.
- The machine language subroutines beginning at the head address specified in [variable 2], which exists in the channel specified in [variable 1], are called and controlled according to the set variables in (D) and (E) registers and (B) and (C) registers in [variable 3] and [variable 4].

NOTE

1. Since the CALL command is a function, use it in the assignment statement format and IF statement format.

```
A=CALL($*, $*****,$*****,$*****)
```

```
IF 1 =CALL($*, $*****,$*****,$***** ) GOTO 200
```

2. For system subroutines which can be called by the CALL command, see Section 4.

INKEY

PRG

FUNCTION

Receives one character of data entered through the keyboard and gives its code to a variable.

FORM

INKEY

EXPLANATION

- Displays the cursor at the specified position and waits for the entry of one character through the keyboard.
- When one character is received, displays it at the aforementioned cursor position and executes the statement of the next line.
- Used to perform the departure operation according to the entered key code.
- As the keyboard, specify the input console which has been set during "console setting". When it is desired to change the keyboard, use the ZIDV command (see P.142).

NOTE

1. During data entry through the keyboard, only the data entered by pressing the **CONT**/**C** keys are used for control of OS side and data entered by pressing other keys are converted into key codes and given to variables. However, the **CONT**/**C** keys cannot be used during run of multi task. (The KD51 depends on the setting of multi task.)
2. Since the INKEY command is a function, use it in the assignment statement format and IF statement format.
A=INKEY
IF \$31=INKEY GOTO 200

EXAMPLE

```
OK
>LIST
 100 REM "EVERCISE"
 110 PRINT,"UNDERSTAND, OR NOT?"
 120 A=INKEY.....Key code of character entered through keyboard is
                    assigned to variable A. It is also possible to omit line
                    number 120 and change line number 130 to IF $43 #
                    INKEY.....
 130 IF A#$43 PRINT*A;GOTO100
                    ..... When C key is pressed, line number 140 is executed.
 140 PRINT*A;PRINT "GOOD"
 150 END

OK
>RUN
UNDERSTAND, OR NOT? A.....Key code of A is $41.
UNDERSTAND, OR NOT? B.....Key code of B is $42.
UNDERSTAND, OR NOT? C.....Key code of C is $43.
GOOD!
OK
>
```

PEEK

FUNCTION

Reads the content of specified memory address.

FORM

PEEK (integer expression)

EXPLANATION

- Reads the content of memory (one byte) in address indicated in [integer expression] in the same channel as the program.
- The range of memory addresses indicated in [integer expression] is addresses 0 to 65535 (\$0 to \$FFFF). However, since 32768 (\$8000) and larger numbers are negative values, specify these numbers in hexadecimal notation and provide the "\$" mark.
- The POKE command (see P.117) has the reverse functions to the PEEK command.

NOTE

Since the PEEK command is a function, use it in the assignment statement format or IF statement format.

```
A=PEEK ($C000)
IF $2B=PEEK ($C000) GOTO 200
```

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE 1"
110 A=$E000 } .....200 is written to address E000.
120 A:0)=200 } .....
130 B=PEEK ($E000) .....Content of address E000 is read to variable B.
140 PRINT B
150 END

OK
>RUN
200 .....Display by line number 140.

OK
>
```

EXAMPLE 2

```
OK
>LIST
100 REM "EXERCISE 2"
110 FOR A=0 TO 100
120 IF $0D=PEEK (A) GOTO 150 ..... When there is data of $0D in address 0 through 100,
130 NEXT A ..... execution jumps to line number 150.
140 END
150 PRINT "0D= ", A
160 END

OK
>RUN
0D=85
OK
>
```

SIZE

TYPE | A

PRG

FUNCTION

Gives the size of used memory area.

FORM

SIZE

EXPLANATION

- Indicates the size of memory area used by the program and shows how far the program can be expanded.
- When multi task is set after the completion of programming with BASIC, the size of memory area used by the program is required. Therefore, be sure to display and record the size of used memory area at the completion of programming.
- The capacity of program can also be calculated by the following table.

Item		Number of Bytes
Line number		2
Program command (excluding GOTO and GOSUB)		1
GOTO, GOSUB	GOTO, GOSUB	3
	Line number	2
ASCII character (excluding small alphabet)		1
Non-ASCII character		2
[CR] Key		1

NOTE

Since the SIZE command is a function, use it in the assignment statement format.

A = SIZE

EXAMPLE 1

```

100 PRINT "MITSUBISHI";GOTO 200 [CR]
 2   1 1111111111111111111 3   2   1
                                     byte
    
```

EXAMPLE 2

```
OK
>LIST
100 INPUT A
110 B=A*A
120 PRINT B
130 END
```

```
OK
>A=SIZE.....Size of memory area used by program is assigned to
OK variable A.
>PRINT $A
0017.....Size of above memory area used by program is
OK displayed.
>
```


SIZE

TYPE B

PRG

FUNCTION

Gives the size of used memory area.

FORM

SIZE

EXPLANATION

- Indicates the size of memory area used by the program and shows how far the program can be expanded.
- When multi task is set after the completion of programming with BASIC, the size of memory area used by the program is required. Therefore, be sure to display and record the size of used memory area at the completion of programming.
- The capacity of program can also be calculated by the following table.

Item		Number of Bytes
Line number		2
Program command (excluding GOTO and GOSUB)		1
GOTO, GOSUB	GOTO, GOSUB	3
	Line number	2
ASCII character		1
CR Key		1

NOTE

Since the SIZE command is a function, use it in the assignment statement format.

A = SIZE

EXAMPLE 1

```

100 PRINT "MITSUBISHI";GOTO 200 CR
    1 11111111111111111111 3 2 1
    byte
  
```

EXAMPLE 2

```
OK
>LIST
 100 INPUT A
 110 B=A*A
 120 PRINT B
 130 END
OK
>A=SIZE.....Size of memory area used by program is assigned to variable A.
OK
>PRINT $A
0017.....Size of above memory area used by program is displayed.
OK
>
```

3.3 Z Commands

- The Z commands are program commands unique to KGPC-BASIC.
- The Z commands include those which are functions and those which are not functions.
- There are the following Z commands.

Z Command	
Those which are not functions	Function
ZCOFF	ZDSP
ZCON	ZMOV
ZCRV	ZRD1
ZDATE	ZRD2
ZIDV	ZWR1
ZNOR	ZWR2
ZODV	
ZTIME	

3.3.1 Z commands which are not functions

- The formats of Z commands, which are not functions, are as follows:

100 ZCON
to
200 ZCOFF

- The Z commands which are not functions are as follows

Command Name	Reference Page
ZCOFF	136
ZCON	138
ZCRV	140
ZDATE	141
ZIDV	142
ZNOR	146
ZDOV	147
ZTIME	151

ZCOFF

TYPE A

PRG

FUNCTION

Erases the cursor.

FORM

ZCOFF

EXPLANATION

- When an instruction to move the cursor is executed after the execution of ZOFF command, the cursor is erased from the CRT screen.
- When program run is at stop, the cursor is displayed.
- The execution of ZCON command (see P.138) displays the cursor even during run of program.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 COLOR 1
 130 ZCOFF
 140 LOCATE 10, 20
 150 PRINT "CURSOR"
 160 GOTO 120
 170 END
```

```
OK
>RUN
```

CURSOR.....Characters in line number 150 are displayed at position specified in line number 140 in color specified in line number 120.

ZCOFF

TYPE B

PRG

FUNCTION

Displays a line under characters.

FORM

ZCOFF

EXPLANATION

- A line is displayed under characters after the execution of ZCOFF command.
- The execution of ZCON command (see P.136) flickers the cursor and stops the display of line under characters.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 ZCOFF
 130 LOCATE 10,20
 140 PRINT "MITSUBISHI"
 150 END
OK
>RUN
      MITSUBISHI
OK
>-
```

ZCON

TYPE A

PRG

FUNCTION

Displays the cursor.

FORM

ZCON

EXPLANATION

- Displays the cursor, which has been erased by the ZCOFF command (see page 136), again.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 COLOR 1
 130 ZCON
 140 LOCATE 10,20
 150 PRINT "CURSOR"
 160 GOTO 160
 170 END
```

```
OK
>RUN
```

```
CURSOR ■
```

.....Characters "CURSOR" are displayed by line number 150 at position specified in line number 140 in color specified in line number 120, and the cursor is also displayed behind the characters.

ZCON

TYPE	B
------	---

PRG

FUNCTION

Stops the display of line under characters.

FORM

ZCON

EXPLANATION

- The execution of ZCON command stops the display of line under characters which have been displayed by the ZCOFF command (see P.137).

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 CLS
 120 ZCOFF
 130 PRINT "MITSUBISHI"
 140 ZCON
 150 PRINT "ELECTRIC"
 160 END

OK
>RUN
MITSUBISHI ELECTRIC
OK
>-
```


ZCRV

PRG

FUNCTION

Reverses display color.

FORM

ZCRV

EXPLANATION

- After the execution of ZCRV command, the color of character and that of screen are reversed one character after the other.
- The execution of ZNOR command (see P.146) restores the reversed colors.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 ZCRV.....Reversal of display color is specified.
 120 PRINT "ABC"
 125 ZNOR.....Display color is restored.
 130 END
OK
>RUN
ABC
OK
>
```

ZDATE

FUNCTION

Provides year, month, day, hour and minute.

FORM

ZDATE variable 1[, variable 2[, variable 3[, variable 4[variable 5]]]]

EXPLANATION

- Year, month, day, hour and minute are set in this order in [variable 1 through variable 5].
- When [variable 3 through variable 5] are omitted, only year and month are set to variable 1 and variable 2.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 ZDATE A, B, C, D.....Year, month, day and hour are set to variables A, B,
                               C and D.
 120 PRINT "YEAR=" , A
 130 PRINT "MONTH=" , B
 140 PRINT "DAY=" , C
 150 PRINT "HOUR=" , D
 160 END

OK
>RUN
YEAR      85.....Display of year by line number 120.
MONTH     8.....Display of month by line number 130.
DAY       6.....Display of day by line number 140.
HOUR     15.....Display of hour by line number 150.
OK
>
```

ZIDV

TYPE A

PRG

FUNCTION

Changes over the input console.

FORM

ZIDV input console number

EXPLANATION

- The ZIDV command allows the entry of data through a desired keyboard when data are entered through the keyboard by use of the INPUT command (see P.93) or INKEY command (see P.128).
- The console used at the start of BASIC has been set to Type K6KB roll keyboard.
- The input console numbers are as shown in the following table:

Number	Input Console
0	K6KB (roll keyboard)
4	Operation panel keyboard

CAUTION

When the operation panel keyboard is selected as the input console, data cannot be entered by the INPUT command.

EXAMPLE

```

OK
>LIST
 100 REM "EXERCISE"
 110 ZIDV 4.....Designation of data entry line number 120 is specified
 120 A=INKFY.....for operation panel key.
 130 PRINT ?A
 140 GOTO 120
 150 END

OK
>RUN
2 C....."X" of operation panel key has been pressed.
3 4....."T" of operation panel key has been pressed.
2 D....."Y" of operation panel key has been pressed.
3 5....."C" of operation panel key has been pressed.

```

ZIDV

TYPE B

PRG

FUNCTION

Changes over the input console.

FORM

ZIDV \$input console number channel number of K31PST

EXPLANATION

- The ZIDV command allows the entry of data through a desired keyboard when data are entered through the keyboard by use of the INPUT command (see P.94) or INKEY command (see P.128).
- At the start of BASIC, K6KB has been set to the K31PST in channel 0.
- Specify the input console number in hexadecimal two digits, and specify the channel number of K31PST at the number of units and the input channel number at the number of tens. The input console numbers are as shown in the following table:

Number	Input Console
1	Type K6KB program console
4	Type K31OP operation panel
5	RS-232C (1CH) of K31PST
6	RS-232C (2CH) of K31PST

- Specify 1CH or 2CH of RS-232C port of K31PST when a general-purpose terminal is connected. The general-purpose terminal can be used only for ASCII code.

NOTE

Avoid the entry of data through Type K6KB program console during online control.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 ZIDV $42.....Type K310p operation panel of K31PST in channel 2
120 A=INKEY.....is specified.
130 PRINT ?A
140 GOTO 120
150 END
```

```
OK
>RUN
32....."2" of K310P of K31PST in channel 2 has been
pressed.
33....."3" of K310P of K31PST in channel 2 has been
pressed.
34....."4" of K310P of K31PST in channel 2 has been
pressed.
```

FUNCTION

Changes over the input console.

FORM

ZIDV channel number

EXPLANATION

- The ZIDV command allows the entry of data through a desired keyboard when data are entered through the keyboard by use of the INPUT command (see P.94) or INKEY command (see P.128).
- The console used at the start of BASIC has been set to the keyboard which is connected to the channel 0 of RS-232C.
- The channel numbers are as shown in the following table:

Number	Input Console Connection Channel
0	RS-232C channel 0
2	RS-232C channel 1
3	RS-232C channel 2
4	RS-422 channel 3

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
105 OPEN 2, $SE, $37, 4 .....Channel 2 of RS-232C is opened.
110 ZIDV 2 .....Designation of data entry in line number 120 is specified
120 A=INKEY .....for keyboard connected to RS-232C channel 2.
130 PRINT SA
140 GOTO 120
150 END

OK
>RUN
41 ..... "A" of keyboard connected to channel 2 of RS-232C
has been pressed.
42 ..... "B" of keyboard connected to channel 2 of RS-232C
has been pressed.
43 ..... "C" of keyboard connected to channel 2 of RS-232C
has been pressed.
44 ..... "D" of keyboard connected to channel 2 of RS-232C
has been pressed.
```

ZNOR

PRG

FUNCTION

Restores the reversed display color.

FORM

ZNOR

EXPLANATION

- Restores the display colors which have been reversed by the ZCRV command (see P.140).

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 ZCRV
 120 PRINT "MELSEC-", .....Line feed is inhibited.
 130 ZNOR
 140 PRINT "KGPC11"
 150 END
OK
>RUN
MELSEC- KGPC11 .....Characters "MELSEC-" in line number 120 and screen
OK                                     color are reversed by line number 110, reversing
>                                     instruction is restored by line number 130, and "KGPC
                                     11" in line number 140 is displayed.
```

FUNCTION

Changes over the output console.

FORM

ZODV \$output console number channel number of K31PST

EXPLANATION

- The ZODV command allows the output of data to a desired output console when data are output by the following command:

Command Name	Reference Page	Command Name	Reference Page	Command Name	Reference Page
CLS	81	LOCATE	99	ZCRV	140
COLOR	79	PRINT	118	ZDSP	153
LCOPY	96	ZCOFF	136	ZNOR	146
LIST	66	ZCON	138	Graphic	182

- At the start of BASIC, set the output console to the color CRT, which is connected to the K31PST in channel number 0, or the general-purpose terminal, which is connected to the 1CH port of RS-232C, by use of SW1 on the K31PST in channel 0.
- The setting number of ZODV command is indicated in hexadecimal two digits. Specify the channel number of K31PST at the number of units and the output console number shown below at the number of tens. Be sure to provide the "\$" mark at the beginning.

Number	Output Console
1	Color CRT
5	RS-232C 1CH
6	RS-232C 2CH
7	Parallel interface

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 ZODV $13 .....Color CRT connected to K31PST in channel 3 is
 120 GCOLOR (5) .....specified.
 130 FTYPE (1)
 140 CIRCLE (512, 200), 100, F
 150 END

OK
>RUN .....Circle in line number 120 and following line numbers
OK .....is displayed on output console specified in line number
V .....110.
```

ZODV

TYPE	B
------	---

PRG

FUNCTION

Changes over the output console.

FORM

ZODV channel number

EXPLANATION

- The ZODV command allows the output of data to a desired output console when data are output by the following command:

Command Name	Reference Page	Command Name	Reference Page
CLS	82	ZCOFF	137
LIST	67	ZCON	139
LOCATE	100	ZCRV	140
PRINT	121	ZNOR	146

- At the start of BASIC, the output console has been set to the color CRT which is connected to the channel number 0 of RS-232C.
- The channel numbers are as follows:

Number	Output Console	Connection Channel
0	RS-232C	Channel 0
1	RS-232C	Channel 1
2	RS-232C	Channel 2
3	RS-422	Channel 3

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 105 OPEN 1, $58, $37, 5.....Channel 1 of RS-232C is opened.
 110 ZODV 1.....Output console connected to channel 1 of RS-232C is
 120 PRINT "MITSUBISHI"      specified.
 130 END

OK
>RUN....."MITSUBISHI" is displayed on output console connected
OK      to channel 1 of RS-232C.
>
```

ZTIME

FUNCTION

Interrupts execution for a predetermined period of time.

FORM

ZTIME time

EXAMPLE 1

- Stops execution during specified period of time \times 10 ms.
- When the ZTIME command is executed during run of multi task, execution is transferred to another task. When the period of time specified by the ZTIME command elapses and the run of another task is interrupted, execution returns to the original task.

EXAMPLE

```
OK
>LIST
100 REM "EXERCISE"
110 FOR I=0 TO 5
120 X=I*I
130 PRINT X
140 ZTIME 10 .....Execution is interrupted for 100 ms.
150 NEXT I
16 END

OK
>RUN
0
1
4 .....Calculation result of line number 120 is displayed per
9 .....100 ms by line number 130.
16
25

OK
>
```

3.3.2 Z commands which are functions

- The Z command, which is a function, returns the result of execution upon completion of execution.
Normal completion-1
Error.0

- The formats of Z commands, which are functions, are as follows:

Assignment statement format

100 A=ZRD1 (\$3, \$8000, X)

IF statement format

100 IF 0=ZRD1 (\$3, \$8000, X) GOTO* *

- The Z commands, which are functions, are as follows:

Command Name	Reference Page
ZDSP	153
ZMOV	157
ZRD1	159
ZRD2	161
ZWR1	163
ZWR2	165

ZDSP

TYPE A

PRG

FUNCTION

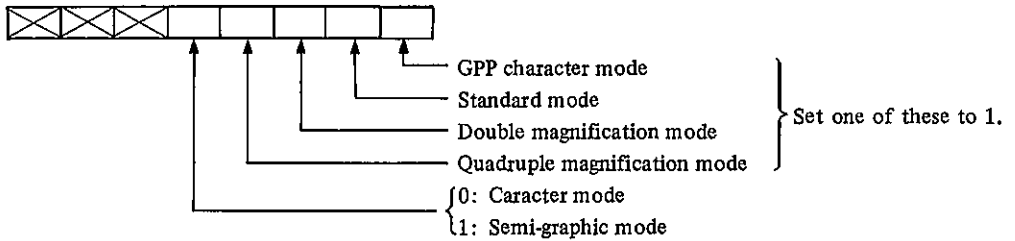
Switches the display mode of CRT.

FORM

ZDSP (mode)

EXPLANATION

- [mode] is one-byte data, and when converted into a binary number, the structure of [mode] is as shown below.



- The character mode and semi-graphic mode are as follows:

	Character Mode	Semi-graphic Mode
GPP character mode	80 x 24 characters, 7 x 9 dots	
Standard mode	36 x 20 characters, 16 x 19 dots	16 x 16 dots
Double magnification mode	18 x 10 characters, 32 x 38 dots	32 x 32 dots
Quadruple magnification mode	9 x 5 characters, 64 x 76 dots	

- The cursor returns to the top left end.
- In the initial state, the display mode is set to the standard mode of the character mode.

NOTE

Since the ZDSP command is a function, use it in the assignment statement format or IF statement format.

A=ZDSP (4)
IF 0=ZDSP (4) GOTO 200

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 CLS
 120 A=ZDSP (4) .....Double mode of character mode is specified.
 130 PRINT "DOUBLE"
 140 END

OK
>RUN
DOUBLE } .....Data is displayed in double mode.
OK
>
```

EXAMPLE 2

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 IF 0=ZDSP (4) GOTO 200 .....When ZDSP command has defect after double mode
 120 PRINT "DOUBLE" .....of character mode is specified, execution jumps to
 130 END .....line number 200 and "ERROR" is displayed.
 200 PRINT "ERROR"
 210 END

OK
>RUN
DOUBLE } .....When the command is normal, data is displayed in
OK .....double mode.
>
```

FUNCTION

Switches the display mode of CRT screen.

FORM

ZDSP (magnification of character screen, magnification of graphic screen)

EXPLANATION

- Switches the sizes of character and graphic displays.
- The sizes of character and graphic displays can be specified separately.
- When the ZDSP command is executed, all data are displayed in the specified size unless the specified sizes are corrected or the initial screen is restored.
- In the initial state, the display mode is set to the standard screen.
- Values specified by the program are as follows:

Specified Value	Content
0	No setting change
1	Standard screen
2	Double magnification screen
4	Quadruple magnification screen

- Data are displayed on the character screen as shown in the following table:

	640 x 400 Dot Mode (20 lines x 40 columns)	1024 x 400 Dot Mode (20 lines x 64 columns)
Standard screen	All of 20 lines x 40 columns are displayed.	All of 20 lines x 64 columns are displayed.
Double magnification screen	Lines 0 to 9 x columns 0 to 19 are displayed in magnified form.	Lines 0 to 9 x columns 0 to 19 are displayed in magnified form.
Quadruple magnification screen	Lines 0 to 4 x columns 0 to 9 are displayed in magnified form.	Lines 0 to 4 x columns 0 to 9 are displayed in magnified form.

- Data are displayed on the graphic screen as shown in the following table:

	640 x 400 Dot Mode	1024 x 400 Dot Mode
Standard screen	All of 640 x 400 dots are displayed.	All of 1024 x 400 dots are displayed.
Double magnification screen	320 x 200 dots from top left of screen are displayed.	512 x 200 dots from top left of screen are displayed.
Quadruple magnification screen	160 x 100 dots from top left of screen are displayed.	256 x 100 dots from top left of screen are displayed.

NOTE

Since the ZDSP command is a function, use it in the assignment statement format or IF statement format.

```
A=ZDSP (2,1)
IF 0=ZDSP (2,1) GOTO 200
```

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 A=ZDSP (2, 1)
 120 PRINT "DOUBLE"
 130 END
OK
>RUN
DOUBLE
OK
>
```

EXAMPLE 2

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 IF 0=ZDSP (2, 1) GOTO 200 .....When ZDSP command has defect after double mode
 120 PRINT "DOUBLE"                of character mode is specified, execution jumps to line
 130 END                            number 200 and "ERROR" is displayed.
 200 PRINT "ERROR"
 210 END
OK
>RUN
DOUBLE
OK
>
```

ZMOV

PRG

FUNCTION

Transfers data in blocks from one memory to another.

FORM

ZMOV (variable 1, variable 2, variable 3, variable 4)

EXPLANATION

- In [variable 1], specify channel numbers in hexadecimal two digits. Specify the channel of transfer destination at the number of units and the channel of transfer source at the number of tens.
- In [variable 2], specify the head address of transfer source.
- In [variable 3], specify the last address of transfer source.
- In [variable 4], specify the head address of transfer destination.
- Between the channels in [variable 1], data stored in the addresses in [variable 2] through [variable 3] are transferred to the addresses which begin at the address in [variable 4].

NOTE

1. The ZMOV command cannot be used for the K3NCPU. (Use SMOV.)
2. Since the ZDSP command is a function, use it in the assignment statement format or IF statement format.
A=ZMOV (\$13, \$8000, \$9FFF, \$8000)
IF 0=ZMOV (\$13, \$8000, \$9FFF, \$8000) GOTO 200
3. Execute the ZMOV command after loading a memory to the channel of transfer destination.

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 A=ZMOV ($51, $8000, $BFFF, $8000)
                                     .....Data stored in addresses 8000 through BFFF of
                                     channel 5 are transferred to address 8000 and
 120 PRINT "TRANSFER COMPLETION" following addresses of channel 1.
 130 END
OK
>RUN
TRANSFER COMPLETION
OK
>
```

EXAMPLE 2

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 IF 0=ZMOV ($1F, $9000, SEFFF, $8000) GOTO 140
                                     .....Data stores 9000 through EFFF of channel 1 are
                                     transferred to address 8000 and following addresses
                                     of channel F.
 120 PRINT "TRANSFER COMPLETION"
 130 END
 140 PRINT "ERROR"
 150 END
OK
>RUN
TRANSFER COMPLETION
OK
>
```

CAUTION

"ERROR" in line number 140 is displayed only in the following cases:

- KGPC1
 - Setting has been done to make communication with SCPU.
 - During communication with the SCPU, transfer is interrupted for 40 msec each time 256 bytes are transferred.
- KD51
 - Communication with KCPU cannot be made.
 - Channel error
 - Channel 4 or 5 has been specified when the extension memory is not loaded.
 - Channel 6, 7, 8 or B to F has been specified.
 - Channel A has been specified for other than K3NCPU.
- Write operation has been attempted to the write inhibit area of KCPU.

ZRD1

PRG

FUNCTION

Read one-byte data.

FORM

ZRD1 (variable 1, variable 2, variable 3)

EXPLANATION

- In [variable 1], specify a channel.
- In [variable 2], specify the memory address of channel specified in [variable 1].
- In [variable 3], specify a variable to which the read data is assigned.
- On-byte data stored in the memory address specified in [variable 2] of channel specified in [variable 1] is assigned to the variable specified in [variable 3].

NOTE

Since the ZRD1 command is a function, use it in the assignment statement format or IF statement format.

```
A=ZRD1 ($3, $8045, X)
IF 0=ZRD1 ($3, $8045, X) GOTO 200
```

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 A=ZRD1 ($3, $9000, X) .....One-byte data in address 9000 of channel 3 is assigned
 120 PRINT $X .....to variable X.
 130 END

OK
>RUN
00C8 .....One-byte data read to variable X in line number 110
OK .....is displayed in hexadecimal four digits by line number
> .....120.
```

EXAMPLE 2

● For KGPC11

```
OK
>LIST
 100 REM "EXERCISE 2"
 110 IF 0=ZRD1 ($EA, $E000, A) GOTO 150
      .....State of input X0, which is stored in address E000 of
      channel EA (area of input X) of K3NCPU, is assigned
      to variable A.
 115 A&1
 120 IF 1=A PRINT "X0=ON"; END
 130 PRINT "X0=OFF"
 140 END
 150 PRINT "ERROR"
 160 END
OK
>RUN
X0=ON .....When state of X0 read by line number 110 is on, it is
OK      assigned to variable A and "X0=ON" is displayed by
>      judgement of line line number 120.
```

CAUTION

1. Address D000 of channel EA stores the ON/OFF data of input X0 of K3NCPU. Since only the lower one bit of the ON/OFF data is effective, mask the other bits.
2. For ZRD1, error results in the following cases:
 - KGPC1
 - Communication cannot be made with SCPU.
 - KGPC11
 - Communication cannot be made with K3NCPU.
 - KD51
 - Communication cannot be made with K3NCPU.
 - Channel error
 - Channel 6, 7, 8, or B to F has been specified.
 - Channel A has been specified for other than K3NCPU.
 - Channel 4 or 5 has been specified when extension memory is not loaded.

ZRD2

FUNCTION

Reads two-byte data.

FORM

ZRD2 (variable 1, variable 2, variable 3)

EXPLANATION

- In [variable 1], specify a channel.
- In [variable 2], specify the memory address of channel specified in [variable 1].
- In [variable 3], specify a variable to which the read data is assigned.
- Two-byte data stored in the memory address specified in [variable 2] of channel specified in [variable 1] is assigned to the variable specified in [variable 3].

NOTE

Since the ZRD2 command is a function, use it in the assignment statement format or IF statement format.

```
A=ZRD2 ($9, $8000, X)
IF 0=ZRD2 ($9, $8000, X) GOTO **
```

EXAMPLE 1

```
OK
>LIST
 100 REM "EXERCISE 1"
 110 A=ZRD2 ($5, $8000, X) .....Two-byte data stored in addresses 8000 and 8001 of
 120 PRINT $X .....channel 5 is assigned to variable X.
 130 END

OK
>RUN
A420 ....."$20" stored in address 8000 of channel 5 and "$A4"
OK .....stored in address 8001 are displayed in hexadecimal
> .....four digits by line number 120.
```

EXAMPLE 2

- For KGPC11

```
OK
>LIST
 100 REM "EXERCISE" 2"
 110 IF 0=ZRD2 ($E2, $E000, X) GOTO 140
 120 PRINT SX
 130 END
 140 PRINT "ERROR"
 150 END
```

Two-byte data stored in DO (addresses E000 and E001) of channel E2 of K3NCPU is assigned to variable X.

```
OK
180A.....Content data register DO read by line number 110 is
OK      displayed in hexadecimal four digits by line number
>      120. High-order (address E001) of DO stores $18, and
      low-order (address E000) stores $0A.
```

CAUTION

For ZRD2, error results in the following cases:

- KGPC1
 - Communication cannot be made with SCPU.
- KGPC11
 - Communication cannot be made with K3NCPU.
- KD51
 - Communication cannot be made with K3NCPU.
 - Channel error
 - Channel 6, 7, 8, or B to F has been specified.
 - Channel A has been specified for other than K3NCPU.
 - Channel 4 or 5 has been specified when extension memory is not loaded.

ZWR1

PRG

FUNCTION

Writes one-byte data.

FORM

ZWR1 (variable 1, variable 2, variable 3)

EXPLANATION

- In [variable 1], specify the channel of write destination.
- In [variable 2], specify the address of write destination of the channel specified in [variable 1].
- In [variable 3], set one-byte data to be written.
- The one-byte data set in [variable 3] is written to the memory address specified in [variable 2] of the channel specified in [variable 1].
- Numeric values which can be set in [variable 3] are 0 to 255.

NOTE

Since the ZWR1 command is a function, use it in the assignment statement format or IF statement format.

A=ZWR1 (\$3, \$8050, \$33)

IF 0=ZWR1 (\$3, \$8050, \$33) GOTO 200

EXAMPLE 1

```
OK
>LIST
100 REM "EXERCISE 1"
110 GOSUB 150
120 A=ZWR1 ($3, $8000, $12) ..... "$12" is written to address 8000 of channel 3.
130 GOSUB 150
140 END
150 B=ZRD1 ($3, $8000, N) } .....Content of address 8000 of channel 3 is displayed
160 PRINT $X
170 RETURN

OK
>RUN
00FF .....Content of address 8000 of channel 3 prior to write in
line number 120.
0012 .....Data written in line number 120 is read by line number
OK .....150 and displayed by line number 160.
>
```


EXAMPLE 2

```

OK
>LIST
100 REM "EXERCISE"
110 GOSUB 170
120 IF 0=ZWR1 ($E2, $E000, S12) GOTO 150
....."$12" is written to address E000 (low-order of data
130 GOSUB 170 .....register DO) of channel E2 of K3NCPU.
140 END
150 PRINT "ERROR"
160 END
170 A=ZRD1 ($E2, $E000, X) } .....Content of address E000 (low-order or data register
180 PRINT ?X .....DO) of channel E2 of K3NCPU is displayed
190 RETURN

OK
>RUN
FF .....Content of low-order of DO before execution of line
number 120.
12 .....Content of low-order of DO after execution of line
number 120.

OK
>

```

CAUTION

"ERROR" in line number 150 is displayed in the following cases:

- KGPC1
 - Communication cannot be made with SCPU.
- KGPC11
 - Communication cannot be made with K3NCPU.
- KD51
 - Communication cannot be made with K3NCPU.
 - Channel error
 - Channel 4 or 5 has been specified when extension memory is not loaded.
 - Channel 6, 7, 8, or B to F has been specified.
 - Channel A has been specified for other than K3NCPU.
- Write to the write inhibit area of KCPU has been attempted.

ZWR2

PRG

FUNCTION

Writes two-byte data.

FORM

ZWR2 (variable 1, variable 2, variable 3)

EXPLANATION

- In [variable 1], specify the channel of write destination.
- In [variable 2], specify the address of write destination of the channel specified in [variable 1].
- In [variable 3], set two-byte data to be written.
- The two-byte data set in [variable 3] is written to the memory address specified in [variable 2] of the channel specified in [variable 1].
- Numeric values which can be set in [variable 3] are -32767 to 32767.

NOTE

Since the ZWR2 command is a function, use it in the assignment statement format or IF statement format.

```

A=ZWR2 ($3, $8000, $1234)
IF 0=ZWR1 ($3, $8000, $1234) GOTO 200

```

EXAMPLE 1

```

OK
>LIST
100 REM "EXERCISE 1"
110 GOSUB 150
120 A=ZWR2 ($1, $9000, $1234) ..... "$34" is written to address 9000 of channel 1 and "$1
2" is written to address 9001.

130 GOSUB 150
140 END
150 B=ZRD2 ($1, $9000, X) } ..... Contents of addresses 9000 and 9001 of channel 1 are
160 PRINT $X } ..... displayed.
170 RETURN

OK
>RUN
039C ..... Contents of addresses 9000 and 9001 of channel 1
before execution of line number 120.
1234 ..... Contents of addresses 9000 and 9001 of channel 1
after execution of line number 120.
OK
>

```

EXAMPLE 2

```

OK
>LIST
100 REM "EXERCISE 2"
110 GOSUB 170
120 IF 0=ZWR2 ($E2, $E000, $1234) GOTO 150
                                     ....."$34" is written to address E000 (low-order of DO)
                                     of channel E2 of K3NCPU and "$12" is written to
                                     address E001 (high-order or DO).

130 GOSUB 170
140 END
150 PRINT "ERROR"
160 END
170 A=ZRD2 ($E2, $E000, X) } .....Contents of addresses E000 and E001 (data register
180 PRINT $X                } .....DO) of channel E2 of K3NCPU are displayed.
190 END

OK
>RUN
0101 .....Contents of data register DO before execution of line
                                     number 120.
1234 .....Contents of data register DO after execution of line
                                     number 120.
OK
>

```

CAUTION

"ERROR" in line number 150 is displayed in the following cases:

- KGPC1
 - Communication cannot be made with SCPU.
- KGPC11
 - Communication cannot be made with K3NCPU.
- KD51
 - Communication cannot be made with K3NCPU.
 - Channel error
 - Channel 4 or 5 has been specified when extension memory is not loaded.
 - Channel 6, 7, 8, or B to F has been specified.
 - Channel A has been specified for other than K3NCPU.
 - Write to the write inhibit area of KCPU has been attempted.

3.4 Built-in Functions

- The built-in functions can be used anywhere in the program only by specifying a command, without specifying a channel and address as in the system subroutines.
- When specifying the argument, which is transferred to the built-in function, always enclose it with parentheses.
- If a real value is specified as an argument instead of an integer, the fraction part is truncated, and only the integer part is used.
- The built-in function is used in the assignment statement format as shown below:

```
A=ABS (-20)
A[0]=ABF (-3.14)
```

- Since the calculated values of commands with * mark are real type, use these commands in the assignment statement format of four-byte indirect variable.
- The built-in functions are as follows:

Command Name	Reference Page
ABF	168
ABS	169
ACOS	170
ASIN	171
ATAN	172
COS	173
EXP	174
LN	175
LOG	176
NOT	177
RND	178
SIN	179
SORT	180
TAN	181

ABF*

INT

FUNCTION

Gives the absolute value of real value of mathematical expression.

FORM

ABF (mathematical expression)

EXPLANATION

- When the data of [mathematical expression] is a negative number, converts it into a positive number.
- Since a real number is used for the value of [mathematical expression], use it in the four-byte indirect variable form.

EXAMPLE 1

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A(0)=-3.14159
 130 A(1)=ABF(A(0))
 140 PRINT . A(1)
 150 END
OK
>RUN
0.3141598E 01
OK
>
```

EXAMPLE 2

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A(0)=-3.14159
 130 A(1)=ABF(A(0))
 140 PRINT . (2.5) A(1)
 150 END
OK
3.14159
OK
>
```

.....Cannot be set on KGPC1.

ABS

INT

FUNCTION

Gives the absolute value of integer value of mathematical expression.

FORM

ABS (mathematical expression)

EXPLANATION

- When the data of [mathematical expression] is a negative number, converts it into a positive number.
- The range of [mathematical expression] is -32767 to 32767 .

EXAMPLE

```
OK
>LIST
100 REM " EXERCISE"
110 A=5 ; B=-20
120 C=A*B
130 D=ABS (C)
140 PRINT D
150 END
```

```
OK
>RUN
100
```

```
OK
>
```

ACOS*

INT

FUNCTION

Gives arccosine (\cos^{-1}) of mathematical expression.

FORM

ACOS (mathematical expression)

EXPLANATION

- Gives arccosine (\cos^{-1}) when the unit of [mathematical expression] is radian ($(\pi/180) \times$ angle).
- The calculated value of ACOS command is in the range of 0 to π .
- For [mathematical expression], either of real number or integer can be used.
- Since the calculated value of ACOS command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A[0]=0.5
 130 A[1]=ACOS(A[0])
 140 A[2]=(A[1]*180)/3.14159
 150 PRINT " RAD=" ,.A[1]
 160 PRINT " DEG=" ,.A[2]
 170 END

OK
>RUN
RAD=0.1047197E 01
DEG=0.5999986E 02
OK
>
```

ASIN*

INT

FUNCTION

Gives arcsine (\sin^{-1}) of mathematical expression.

FORM

ASIN (mathematical expression)

EXPLANATION

- Gives arcsine (\sin^{-1}) when the unit of [mathematical expression] is radian ($(\pi/180) \times \text{angle}$).
- For [mathematical expression], either of real number or integer can be used.
- The calculated value of ASIN command is in the range of $-\pi/2$ to $\pi/2$.
- Since the calculated value of ASIN command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 A=$E000
 120 A[0]=ASIN(0.5)
 130 A[1]=(A[0]*180)/3.14159
 140 PRINT "RAD=" ..A[0]
 150 PRINT "DEG=" ..A[1]
 160 END

OK
>RUN
RAD=0.5235987E 00
DEG=0.29999994E 02
OK
>
```


ATAN*

INT

FUNCTION

Gives arctangent (\tan^{-1}) of mathematical expression.

FORM

ATAN (mathematical expression)

EXPLANATION

- Gives arctangent (\tan^{-1}) when the unit of [mathematical expression] is radian ($(\pi/180) \times$ angle).
- The calculated value of ATAN command is in the range of $-\pi/2$ to $\pi/2$.
- Since the calculated value of ATAN command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A{0}=ATAN(1)
 130 A{1}=(A{0}*180)/3.14159
 140 PRINT " RAD=" ,.A{0}
 150 PRINT " DEG=" ,.A{1}
 160 END

OK
>RUN
RAD=0.7853980E 00
DEG=0.4499989E 02
OK
>
```

COS*

INT

FUNCTION

Gives the value of cosine (cos) of mathematical expression.

FORM

COS (mathematical expression)

EXPLANATION

- Gives cosine (cos) when the unit of [mathematical expression] is radian ($(\pi/180) \times \text{angle}$).
- For [mathematical expression], either of real number or integer can be used.
- Since the calculated value of COS command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
100 REM " EXERCISE "
110 A=$E000
120 A[0]=COS(30/180*3.14159)
130 PRINT " COS=" , .A[0]
140 END
OK
>RUN
COS=0.8660247E 00
OK
>
```

EXP*

PRG

FUNCTION

Gives the value of exponential function with respect to e ($e = 2.718281$).

FORM

EXP (mathematical expression)

EXPLANATION

- Gives the value of exponential function (e [mathematical expression]) with respect to e when the value of [mathematical expression] is an exponent.
- The range of value of [mathematical expression] is -31.99990 to 31.99990 .
- When the value calculated by the EXP command overflows, the command returns "0" and then continues the run of program.
- Since the calculated value of EXP command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM "EXERCISE"
 110 A=$E000
 120 FOR I=0 TO 5
 130 A[I]=EXP(I)
 140 PRINT,A[I]
 150 NEXT I
 160 END
```

```
OK
>RUN
0.10000000E 01
0.2718281E 01
0.7389056E 01
0.2008553E 02
0.5459815E 02
0.1484131E 03
OK
>
```

LN***FUNCTION**

Gives the value of natural logarithm (loge).

FORM

LN (mathematical expression)

EXPLANATION

- Gives the natural logarithm (loge) of the value of [mathematical expression].
- In [mathematical expression], specify a positive number.
- Since the calculated value of LN command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A(0)=2.718282
 130 A(1)=LN(A(0))
 140 PRINT.A(1)
 150 END
OK
>RUN
1.00000001E 00
OK
>
```

LOG***FUNCTION**

Gives the value of common logarithm (\log_{10}).

FORM

LOG (mathematical expression)

EXPLANATION

- Gives the common logarithm (\log_{10}) of the value of [mathematical expression].
- In [mathematical expression], specify a positive number.
- Since the calculated value of LOG command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A[0]=LOG(10)
 130 PRINT.A[0]
 140 END
OK
>RUN
0.99999997E 00
OK
>
```

NOT

FUNCTION

Gives "1" when the value of data is "0", and gives "0" when the value is not "0".

FORM

NOT (mathematical expression)

EXPLANATION

Gives "1" when the value of expression is "0", and gives "0" when the value is not "0".

EXAMPLE 1

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=ZRD1 (3, $9826, X)
 120 B=NOT (X)
 130 PRINT X
 140 PRINT B
 150 END

OK
>RUN
0.10000000E01
0.00000000E01

OK
>
```

RND

INT

FUNCTION

Generates random numbers in the range of 1 and specified value.

FORM

RND (mathematical expression)

EXPLANATION

- Gives random numbers in the range of 1 and the value specified by mathematical expression.
- For the generated random numbers, only integer can be specified.
- In [mathematical expression], only integer can be specified.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 FOR I=100 TO 105
 120 X=RND (I)
 130 PRINT X
 140 NEXT I
 150 END
OK
>RUN
 88
 69
 75
 84
 87
 50
OK
>
```

SIN*

INT

FUNCTION

Gives the value of sine (sin) of mathematical expression.

FORM

SIN (mathematical expression)

EXPLANATION

- Gives sine (sin) when the unit of [mathematical expression] is radian ($(\pi/180) \times \text{angle}$).
- For [mathematical expression], either of real number or integer can be used.
- Since the calculated value of SIN command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A [0] =SIN (60/180*3.14159)
 130 PRINT .A [0]
 140 END
OK
>RUN
0.8660261E 00
OK
>
```


SQRT*

INT

FUNCTION

Gives the square root of the value of mathematical expression.

FORM

SQRT (mathematical expression)

EXPLANATION

- Gives the square root of the value of mathematical expression.
- The value of [mathematical expression] must be "0" or a positive number.
- Since the calculated value of SQRT command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
100 REM " EXERCISE"
110 FOR I=0 TO 5
120 A=$E000
130 A (0) =SQRT (I)
140 PRINT "SQUARE ROOT OF",#2,I,"=",.A(0)
150 NEXT I
160 END
```

```
OK
>RUN
SQUARE ROOT OF 0=0.00000000E 01
SQUARE ROOT OF 1=0.10000000E 01
SQUARE ROOT OF 2=0.1414213E 01
SQUARE ROOT OF 3=0.1732050E 01
SQUARE ROOT OF 4=0.2000000E 01
SQUARE ROOT OF 5=0.2236067E 01
```

```
OK
>
```

NOTE

If the SQRT command is executed when the value of mathematical expression is negative, the calculation result is 0.

TAN*

INT

FUNCTION

Gives tangent (tan) of the value of mathematical expression.

FORM

TAN (mathematical expression)

EXPLANATION

- Gives tangent (tan) obtained when the unit of [mathematical expression] is radian ($(\pi/180)$ x angle).
- For the value of [mathematical expression], either of real number or integer can be used.
- Since the calculated value of TAN command is real type, use it in the four-byte indirect variable form.

EXAMPLE

```
OK
>LIST
 100 REM " EXERCISE"
 110 A=$E000
 120 A(0)=TAN(60/180*3.14159)
 130 PRINT.A(0)
 140 END
OK
>RUN
 0.1732058E 01
OK
>
```

3.5 Graphic Commands

- The graphic commands are used to draw a figure, such as a line, circle and quadrangle, and to specify the display area of color CRT.
- The graphic commands are effective only for the color CRT.
- The graphic commands are as follows:

Command Name	Reference Page
CIRCLE	183
FSTYLE	185
FTYPE	187
GCOLOR	189
LINE	191
LTYPE	193
MODE	195
ORIGIN	201
PRESET	204
PSET	207
VIEW	208
WINDOW	211

CIRCLE

GRF

FUNCTION

Draws a circle or a sector.

FORM

CIRCLE (X coordinate, Y coordinate), radius[,F]
CIRCLE (X coordinate, Y coordinate), radius, (angle 1, angle 2)[,F]

EXPLANATION

- In [X coordinate, Y coordinate], specify the center of circle with the number of dots.
- In r[radius], specify the radius of circle to be drawn with the number of dots.
- [angle 1, angle 2] is used to draw a sector. Specify angles.
- The sector is drawn in the counterclockwise direction from the angle specified in [angle 1] to the angle specified in [angle 2].
- When the same value is specified in [angle 1] and [angle 2], a circle is drawn.
- When an angle of 360° or more or a negative number is specified in [angle 1] and/or [angle 2], the specified value is divided by 360 and its remainder is applied to the angle.
- When a value is specified in [F], the inside of circle or sector is painted out in the form specified by the FTYPE command (see page 187).
- Specify the color by use of the GCOLOR command (see page 189).

NOTE

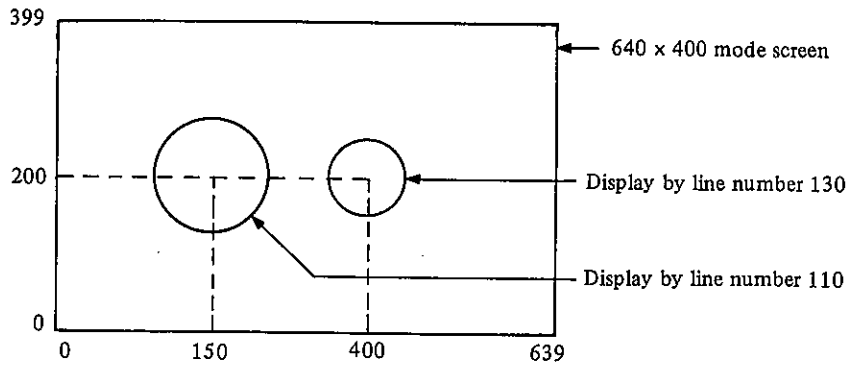
1. When the 20-inch color CRT is used, if the 1024×400 mode is selected for the color CRT in the setting of initial screen setting device, a true circle is not drawn but an ellipse is drawn. When it is desired to draw a circle, select the 640×400 mode.
2. The the bottom left of screen is the reference position (0, 0).

EXAMPLE 1

```

OK
>LIST
100 REM " EXERCISE 1"
110 CIRCLE (150, 200) , 75 .....A circle of 75-dot radius is drawn, centered at position
120 FTYPE (1)                    of X coordinate 150 and Y coordinate 200.
130 CIRCLE (400, 200) , 50 , F.....A circle of 50-dot radius is drawn, centered at position
140 END                          of X coordinate 400 and Y coordinate 200, and the in-
                                  side of circle is painted.
OK
>RUN

```

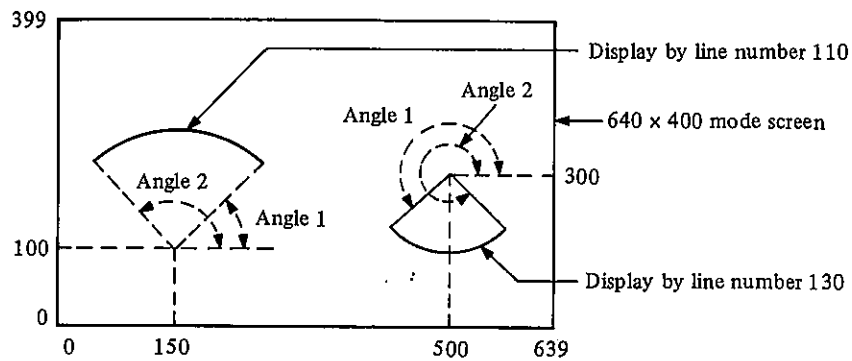


EXAMPLE 2

```

OK
>LIST
100 REM " EXERCISE 2"
110 CIRCLE (150, 100) , 150 , (45 , 135) ..... Only 45° and 135° range of circle of 150-dot radius
120 FTYPE (1)                    centered at X coordinate 150 and Y coordinate 100 is
                                  displayed.
130 CIRCLE (500, 300) , 100 , (225 , 315) , F..... Only 225° and 315° range of circle of 100-dot radius
140 END                          centered at X coordinate 500 and Y coordinate 300 is
                                  displayed, and the inside of sector is
                                  painted.
OK
>RUN

```



FSTYLE

FUNCTION








Specifies the painting details of FTYPE command.

FORM

FSTYLE (number)

EXPLANATION

- Used only when number 2 or 3 of the FTYPE command (see page 187) is specified.
- When FTYPE(2) has been specified, the setting of ASCII code to the number of FSTYLE command in hexadecimal notation causes the inside of figure to be painted by characters corresponding to the code. ASCII codes which can be specified are \$20 to \$5F.
- When FTYPE(3) has been specified, set one of the numbers shown in the following table to the number of FSTYLE command.

Number	Painting Form
0 or 7	 [Same as FTYPE(1)]
1	
2	
3	
4	
5	
6	

CAUTION

The FSTYLE command cannot be used for the CIRCLE command (see page 183).

EXAMPLE 1

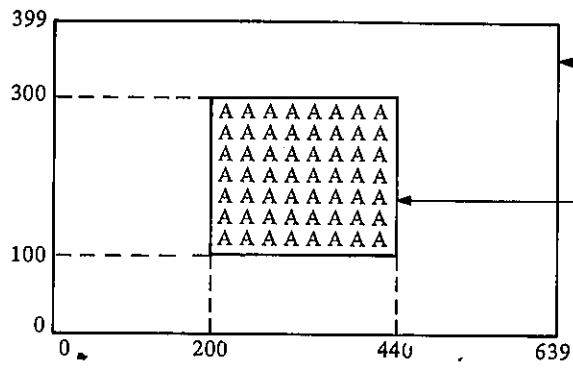
FTYPE(2) has been specified

```

OK
LIST
100 REM " EXERCISE 1"
110 FTYPE (2)
120 FSTYLE ($41) .....
130 LINE (200,100) - (440,300) ,BF .....
140 END
OK
>RUN

```

Specified area is painted with character corresponding to key code \$41.
A rectangle, which has diagonal points at point of X coordinate 200 and Y coordinate 100 and point of X coordinate 440 and Y coordinate 300, is drawn and the inside is painted in the form of line number 120.



← 640 x 400 mode screen
← The inside of rectangle specified in line number 130 is painted with character A specified in line number 120.

EXAMPLE 2

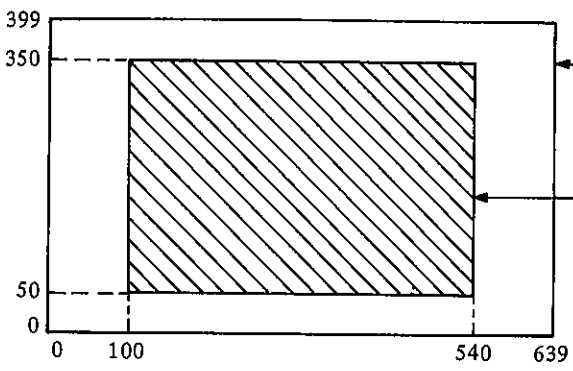
FTYPE(3) has been specified

```

OK
>LIST
100 REM " EXERCISE 2"
110 FTYPE (3)
120 FSTYLE (2) .....
130 LINE (100,50) - (540,350) ,BF .....
140 END
OK
>RUN

```

Form of hatching is specified.



← 640 x 400 mode screen
← The inside of rectangle specified in line number 130 is painted in the form specified in line number 120.

FTYPE

FUNCTION

Paints the specified area.

FORM

FTYPE (number)

EXPLANATION

- Specifies the type of painting of the area specified by the graphic command.
- The types of painting are as shown in the following table.

Number	Painting Type
0	Only a peripheral line is drawn and the inside is not painted out.
1	The area is completely painted.
2	The area is painted by characters corresponding to the ASCII code specified by the FSTYLE command.
3	The area is painted by the hatching specified by the FSTYLE command.

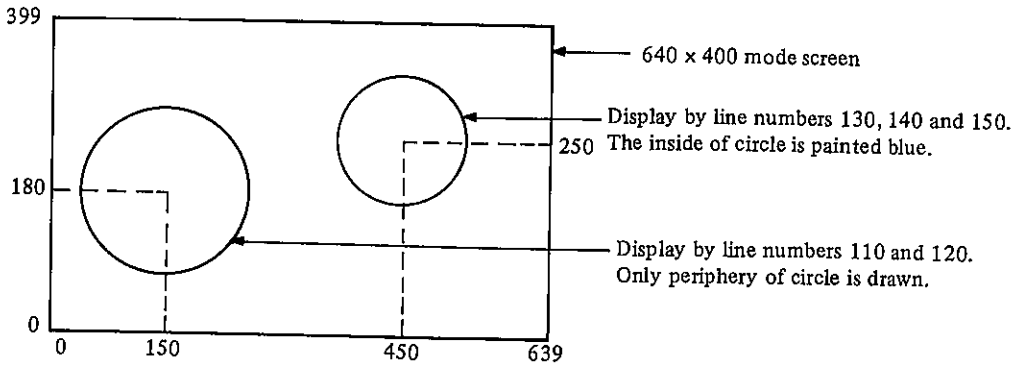
- For the numbers 2 and 3, see the FSTYLE command (see page 183),
- To specify a color, specify the GCOLOR command (see page 189).

CAUTION

When the inside of a circle is painted by the CIRCLE command (see page 183), the numbers 2 and 3 of FTYPE command cannot be used. When the number 2 or 3 is specified, the same type as that of number 1 is set.

EXAMPLE

```
OK
>LIST
100 REM " EXERCISE"
110 FTYPE (0)
120 CIRCLE (150,180) ,100 , F } ..... Only periphery of circle is drawn.
130 FTYPE (1) } ..... Line number 110 can be omitted.
140 GCOLOR (1) } ..... The inside of circle is completely painted with color
150 CIRCLE (450,250) ,80 , F } ..... specified in line number 140.
160 END
OK
>RUN
```



GCOLOR

FUNCTION

Specifies the color of graphic screen.

FORM

GCOLOR (color code)

EXPLANATION

- Specifies the color of figure when the figure is drawn by the graphic command.
- The color codes are as shown in the following table (the same as those of the COLOR command (see page 79)).

Color Code	Specified Color	Color Code	Specified Color
0	Black	4	Green
1	Blue	5	Light blue
2	Red	6	Yellow
3	Purple	7	White

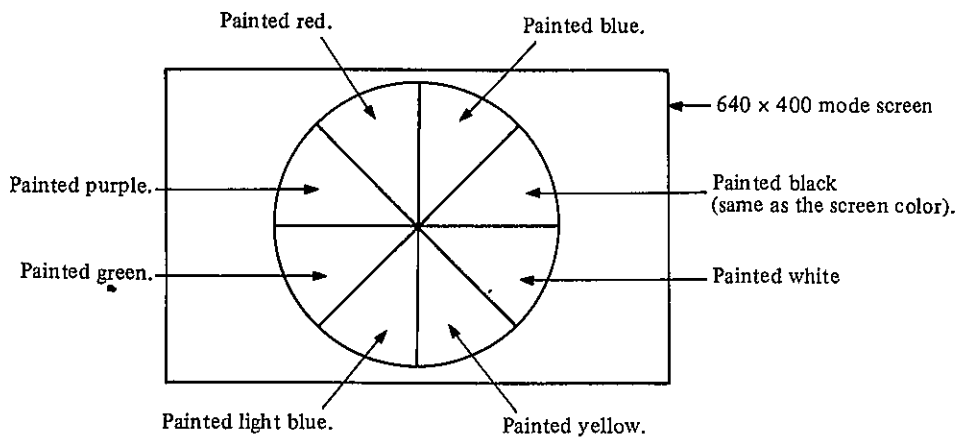
- When the GCOLOR command is not specified, the color is set to white.
- To specify the color of character on the text screen, use the COLOR command.

EXAMPLE

```

OK
>LIST
100 REM " EXERCISE"
110 CLS
120 A=0
130 B=0
140 FTYPE (1)
150 GCOLOR (A)
160 CIRCLE (320,200),180,(B,B+45),F }..... Sector of 45° is painted with color
170 A=A+1                                specified in line number 150.
180 B=B+45
190 IF B=360 END
200 GOTO 140
OK
>RUN

```



LINE

FUNCTION

Draws a straight line and rectangle.

FORM

LINE (X1 coordinate, Y1 coordinate)–(X2 coordinate, Y2 coordinate)
[,B[F]]

EXPLANATION

- When [BF] is omitted, a straight line is drawn from the point indicated in (X1, Y1 coordinates) to the point indicated in (X2, Y2 coordinates).
- The type of line, such as a continuous line and broken line, can be selected by the LTYPE command (see page 193). when the LTYPE command is not specified, the line is set to the continuous line.
- When only [B] is specified, only the periphery of a rectangle, which has a diagonal line between the point indicated in (X1, Y1 coordinated) and the point indicated in (X2, Y2 coordinates), is drawn.
- When [BF] is specified, a rectangle, which has diagonal points at the point indicated in (X1, Y1 coordinates) and the point indicated in (X2, Y2 coordinates), is drawn, and the inside of the rectangle is painted in the form specified by the FTYPE command (see page 187).
- When [BF] is specified, if the FTYPE command is not specified, only the periphery of rectangle is drawn and the inside is not painted out.
- To specify a color, use the GCOLOR command (see page 189).

NOTE

The bottom left of screen is the reference position (0, 0).

EXAMPLE 1

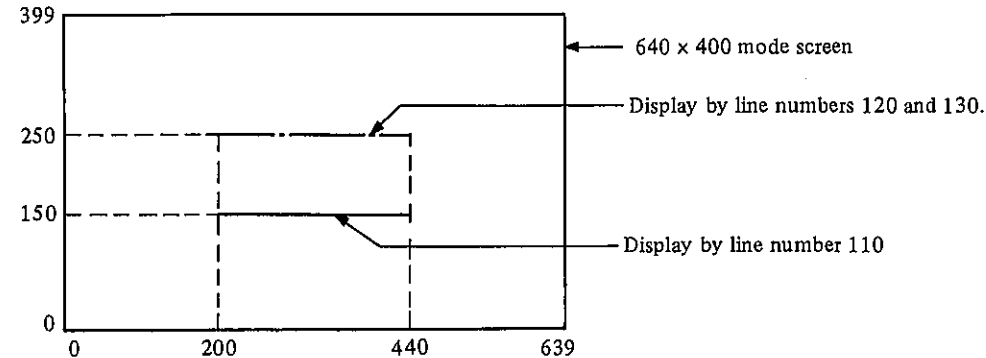
```

OK
>LIST
100 REM " EXERCISE 1"
110 LINE (200,150) - (440,150)
120 LTYPE (4)
130 LINE (200,250) - (440,250)
140 END
OK
>RUN

```

Line from point of X coordinate 200 and Y coordinate 150 to point of X coordinate 440 and Y coordinate 150 is displayed in continuous line.

Line from point of X coordinate 200 and Y coordinate 250 to point of X coordinate 440 and Y coordinate 250 is displayed in the form of line specified in line number 120.



EXAMPLE 2

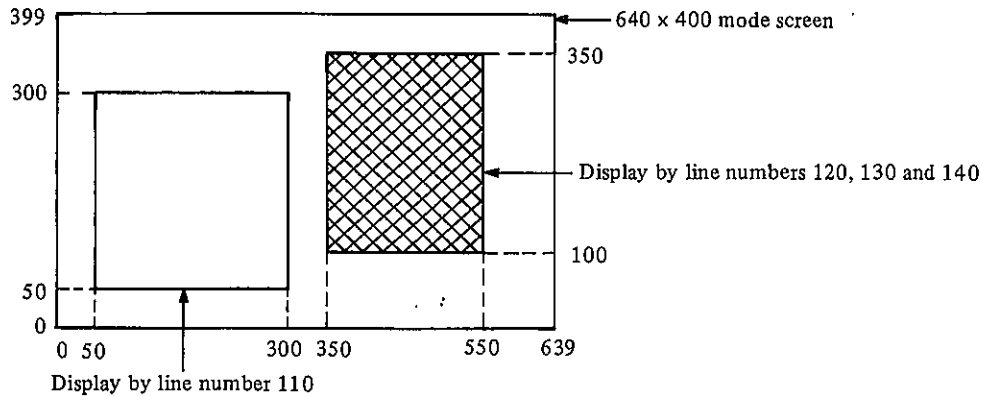
```

OK
>LIST
100 REM " EXERCISE 2"
110 LINE (50,50) - (300,300) , B
120 FTYPE (3)
130 FSTYLE (3)
140 LINE (350,100) - (550,350) , BF
150 END
OK
>RUN

```

Rectangle, which has diagonal points at point of X coordinate 50 and Y coordinate 50 and point of X coordinate 300 and Y coordinate 300, is displayed.

Rectangle, which has diagonal points at point of X coordinate 350 and Y coordinate 100 to point of X coordinate 550 and Y coordinate 350, is displayed and its inside is painted in the form specified in line numbers 120 and 130.



LTYPE

FUNCTION


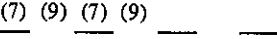

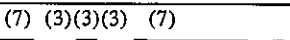
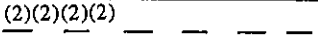

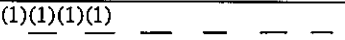
Specifies the type of straight line.

FORM

LTYPE (number)

EXPLANATION

- When drawing a straight line by the LINE command (see page 191), select the type of the straight line.
- Straight lines, which can be handled by the LTYPE command, are as shown in the following table.

Number	Type of Straight Line
1	Continuous line 
2	Broken line 
3	Dotted line 1 
4	Alternate long and short dash line 
5	Dotted line 2 
6	Dotted line 3 
7	Dotted line 4 

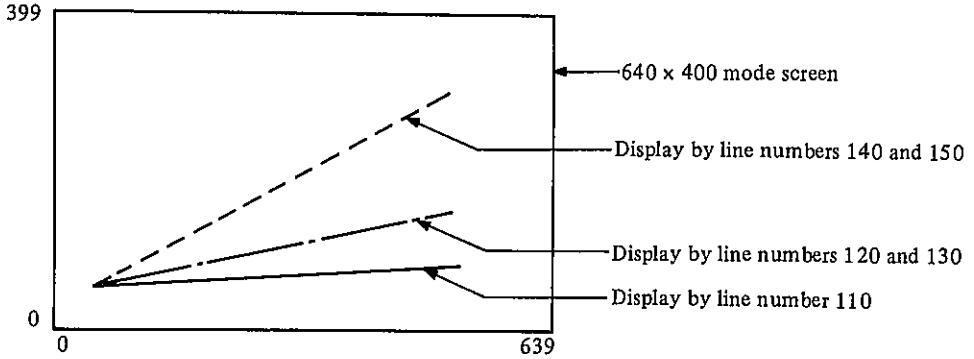
() The number of dots is indicated in parentheses.

- When the number 1 is specified, the LTYPE command can be omitted.

EXAMPLE

```
OK
>LIST
100 REM " EXERCISE"
110 LINE (50,50) - (500,80) .....Line is displayed at specified position in continuous line.
120 LTYPE (2)
130 LINE (50,50) - (500,150) } .....Line in the form specified in line number 120 is display-
140 LTYPE (3)                  ed at the position specified in line number 130.
150 LINE (50,50) - (500,300)
160 END
```

```
OK
>RUN
```



MODE

FUNCTION

Performs operation of the color of two or more figures drawn by the graphic command.

FORM

MODE (number)

EXPLANATION

- Performs operation of the color of overlapping portion of two or more figures drawn by the graphic command.
- The operation methods are as shown in the following table.

Number	Operation Method
1	Assignment
2	Operation by logical sum (OR)
3	Operation by exclusive logical sum (XOR)
4	Erasure of the bit of specified color

- The colors specified by the GCOLOR command comprise the three primary colors of RGB (red, green, blue), and are combined as shown below depending on which bits of RGB are set to "1".

Color	Bit			Color	Bit		
	Green	Red	Blue		Green	Red	Blue
Black	0	0	0	Green	1	0	0
Blue	0	0	1	Light blue	1	0	1
Red	0	1	0	Yellow	1	1	0
Purple	0	1	1	White	1	1	1

- When [number 1] is specified, the color of overlapping portion of figures change to the color specified after the execution of MODE command.

- When [number 2] is specified, logical sum (OR) is calculated in regards to bits of the color of figure specified before the execution of MODE command and the color of figure specified after the execution, and the calculated value is used as the color code of the figure of overlapping portion.

Color of figure drawn before execution of MODE command: 0 0 1 Blue

Color of figure drawn after execution of MODE command: 0 1 0 Red

Color of figure of overlapping portion: 0 1 1 Purple

- When [number 3] is specified, exclusive logical sum (XOR) is calculated in regards to bits of the color of figure specified before the execution of MODE command and the color of figure specified after the execution, and the calculated value is used as the color code of the figure of overlapping portion.

Color of figure drawn before execution of MODE command: 1 0 1 Light blue

Color of figure drawn after execution of MODE command: 0 1 1 Purple

Color of figure of overlapping portion: 1 1 0 Yellow

CAUTION

When the CIRCLE command is used, MODE(3) cannot be used.
--

- When [number 4] is specified, only the color code, among the bits of green, red and blue, used for the color of the figure specified after the execution of MODE command is erased from the color consisting of the bits of green, red and blue of the figure specified before the execution, and the result is used as the color code of the figure of overlapping portion. When the bits unused for the color code before the execution has been specified for the color after the execution, the color code of the corresponding portion after the execution is ignored.

Color of figure drawn before execution of MODE command: 1 1 0 Yellow

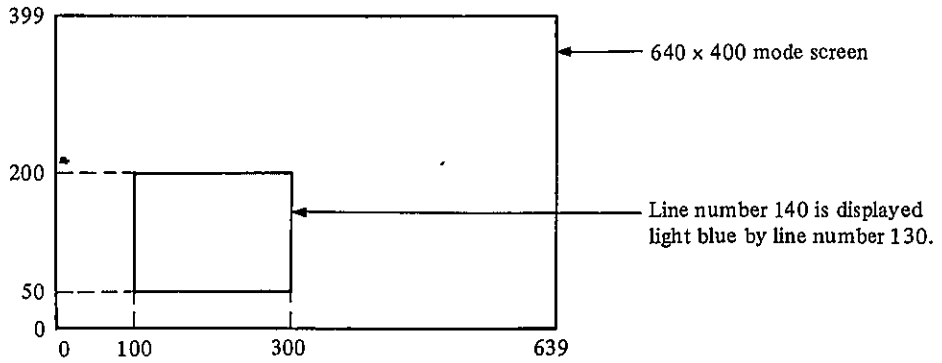
Color of figure drawn after execution of MODE command: 0 1 1 Purple

Color of figure of overlapping portion: 1 0 0 Green

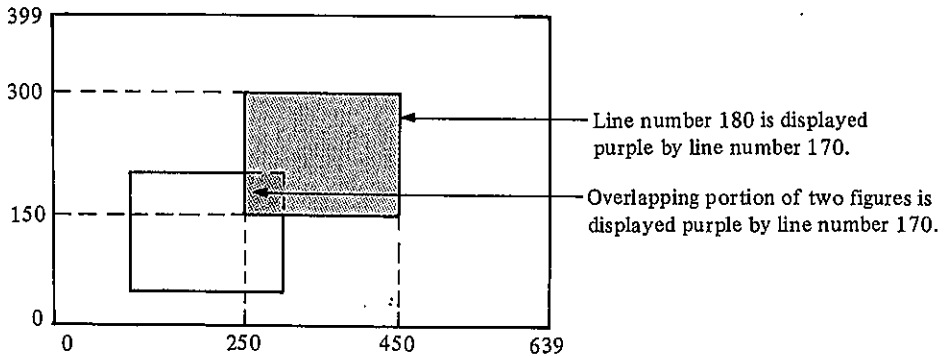
EXAMPLE 1

MODE(1) has been specified

```
OK
>LIST
 100 " EXERCISE 1"
 110 CLS
 120 FTYPE (1)
 130 GCOLOR (5)
 140 LINE (100 , 50) - (300 , 200) ,BF
 150 BREAK
 160 MODE (1)
 170 GCOLOR (3)
 180 LINE (250 , 150) - (450 , 300) ,BF
 190 END
OK
>RUN
```

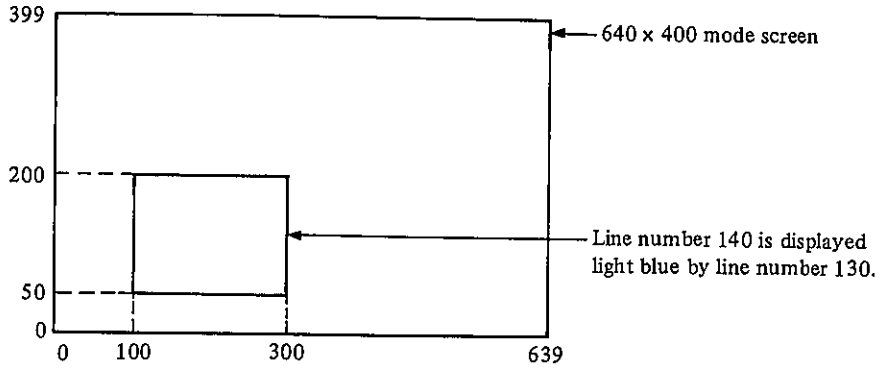


```
OK
>CONT
```

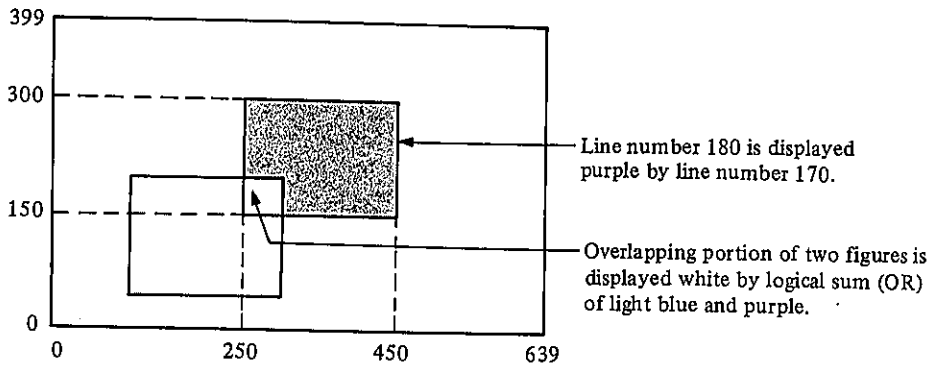


EXAMPLE 2

MODE(2) has been specified
The line number 160 of EXAMPLE 1 is MODE(2)

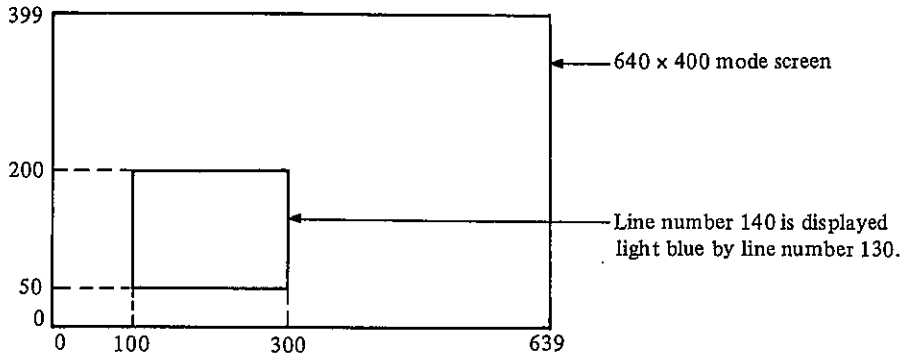


OK
>CONT

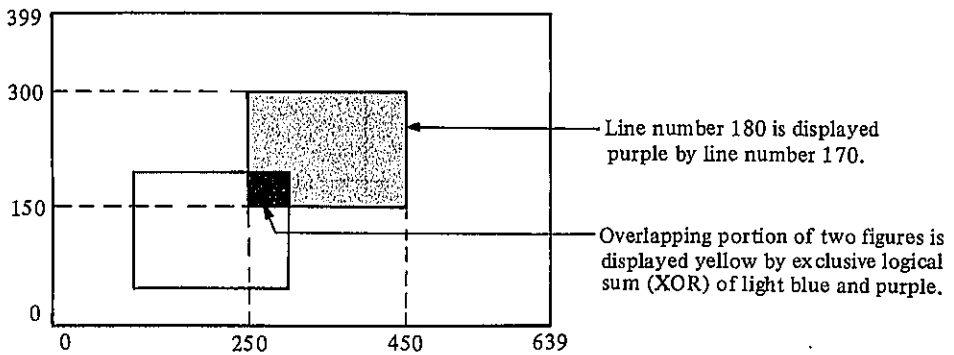


EXAMPLE 3

MODE(3) has been specified
The line number 160 of EXAMPLE 1 is MODE(3)

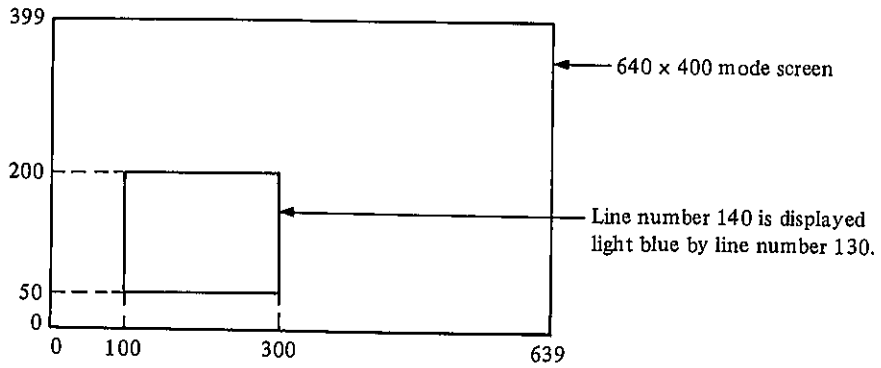


OK
>CONT

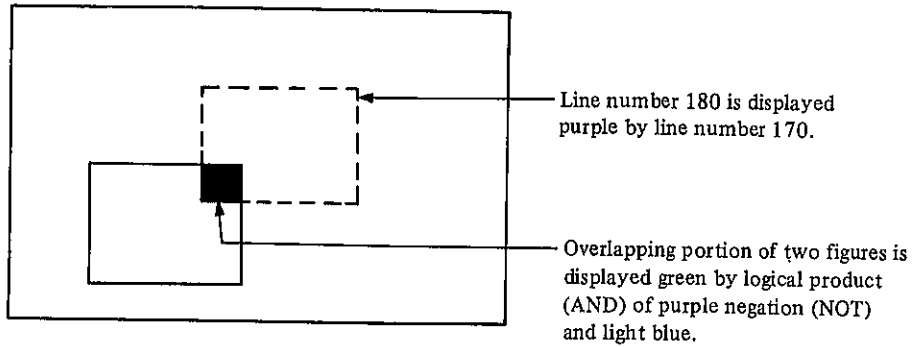


EXAMPLE 4

MODE(4) has been specified
The line number 160 of EXAMPLE 1 is MODE(4)



OK
>CONT



ORIGIN

FUNCTION

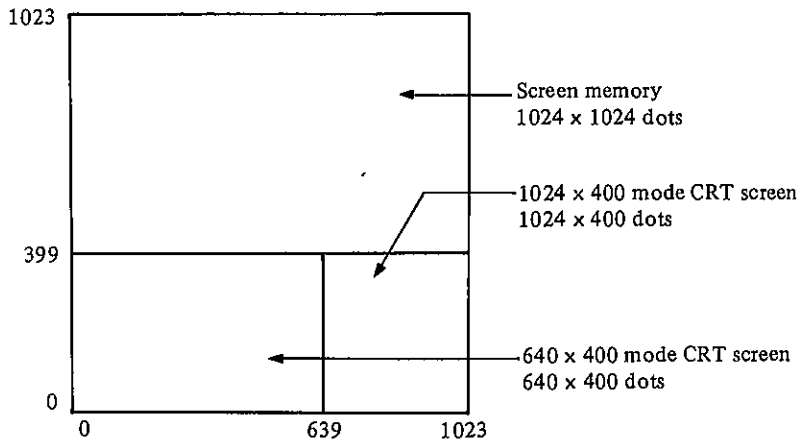
Moves the display of CRT screen to desired location of screen memory.

FORM

ORIGIN (X coordinate, Y coordinate)

EXPLANATION

- Specify which area of 1024 x 1024-dot screen memory is displayed on the CRT screen.
- The initial state is the state after ORIGIN(0, 0) has been executed.
- When a figure has been drawn in the whole area of screen memory, the portion desired to be displayed can be displayed on the CRT screen by the ORIGIN command.
- The relation between the screen memory and the CRT screen is normally as shown below:



NOTE

When moving the display of CRT screen to desired location of screen memory by the ORIGIN command and drawing a figure by the graphic command, the X and Y coordinates of portion displayed on the CRT screen are the same as the coordinates before the execution of ORIGIN command.

EXAMPLE

```

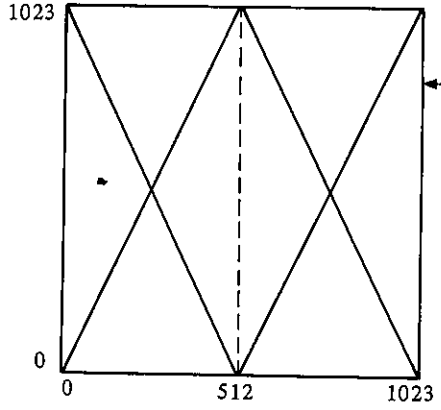
OK
>LIST
 90 CLS
100 REM " EXERCISE"
110 LINE (0,0) - (512,1023)
120 LINE (512,1023) - (1023,0)
130 LINE (0,1023) - (512,0)
140 LINE (512,0) - (1023,1023)
150 ZTIME 500
160 ORIGIN (192,0) .....
170 ZTIME 500
180 ORIGIN (192,312) .....
190 ZTIME 100
200 CIRCLE (320,200),100
210 ZTIME 500
220 ORIGIN (192,624) .....
230 END
  
```

} Figure is drawn by use of whole screen memory area.

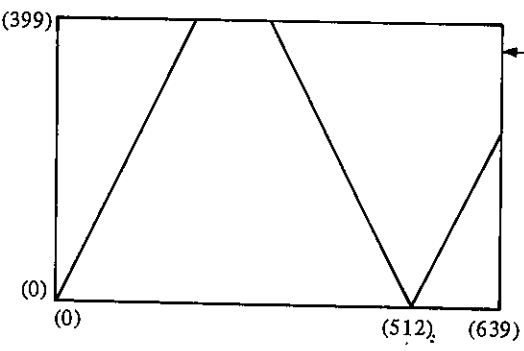
..... Display of CRT screen is transferred to position specified in screen memory.

```

OK
>RUN
  
```



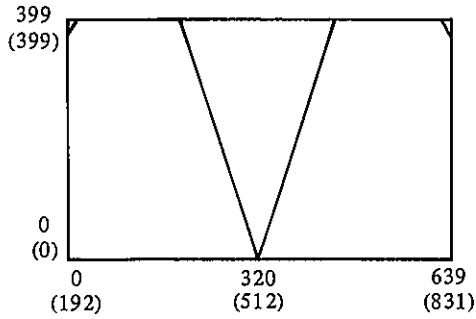
Screen memory
Lines are drawn by line numbers 110 to 140.



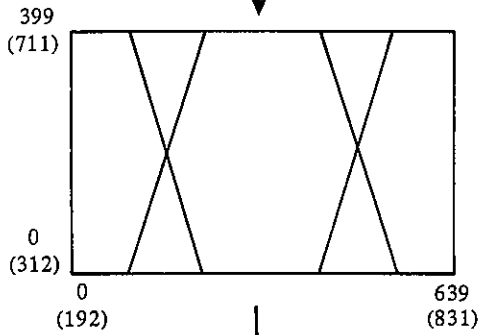
640 x 400 mode screen
Portion of dots (0 to 639) x (0 to 399)
in screen memory is displayed on the
CRT screen.

Coordinates in parentheses indicate
coordinates of screen memory.

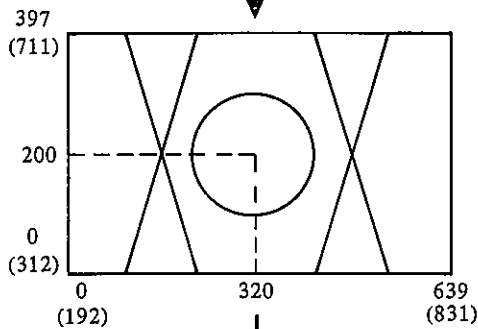




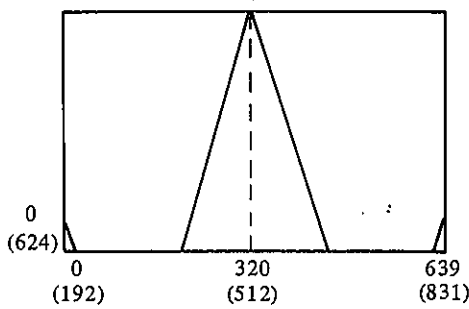
Display by line number 160
 Portion of dots (192 to 831) x (0 to 399) in screen



Display by line number 180
 Portion of dots (192 to 831) x (312 to 711) in screen memory is displayed on the CRT screen.



Circle is displayed by line number 200.
 When figure is drawn by graphic command, specify X and Y coordinates at the coordinates of CRT screen.



Display by line number 220
 Portion of dots (192 to 831) x (624 to 1023) in screen memory is displayed on the CRT screen.

PRESET

GRF

FUNCTION

Erases one desired dot from the graphic screen.

FORM

PRESET (X coordinate, Y coordinate)

EXPLANATION

- Erases one dot which is located at the position specified in [X, Y coordinates].
- By using the PRESET command and the GCOLOR command (see page 189) together, a desired color can be erased.
Namely, since the colors comprise the three primary colors of RGB (red, green, blue), if it is attempted to erase a dot, which is located at the position specified in purple consisting of R (red) and B (blue), by the PRESET command and GCOLOR command from the white portion consisting of R (red), G (green) and B (blue), only R and B are erased and white changes to green.
- The compositions of colors by the GCOLOR command are as shown in the following table:

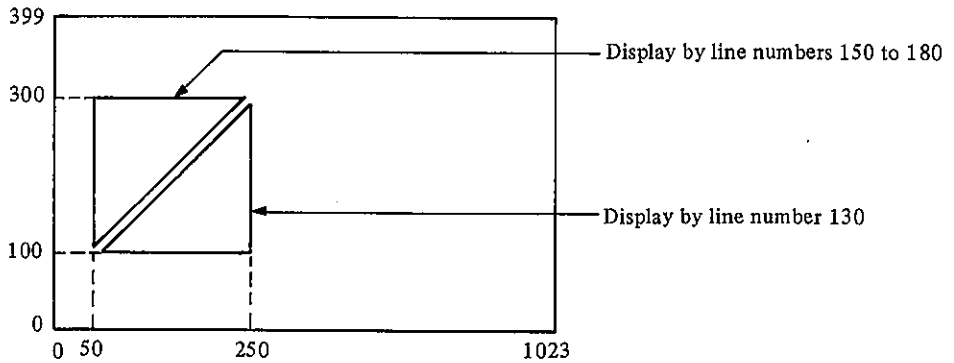
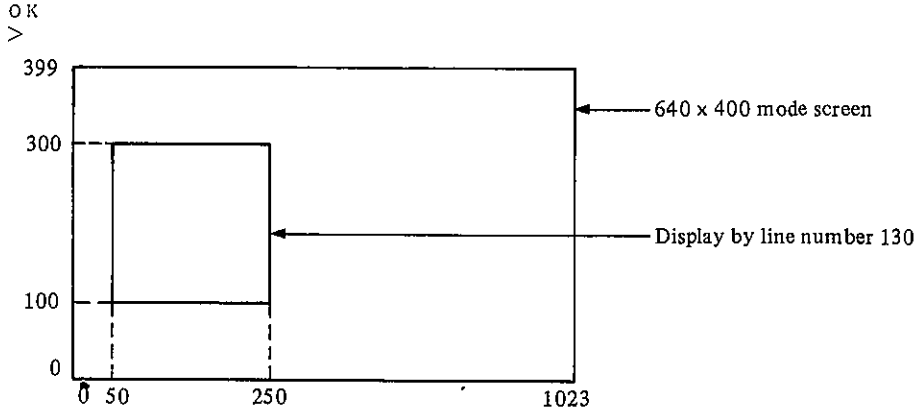
Color	Composition	Color	Composition
Black	—	Green	Green
Blue	Blue	Light blue	Green, blue
Red	Red	Yellow	Green, red
Purple	Blue, red	White	Green, red, blue

EXAMPLE 1

```

OK
>LIST
100 REM " EXERCISE 1"
110 CLS
120 FTYPE (1)
130 LINE (50,100) - (250,300) ,BF
140 A=50
150 FOR B=100 TO 300
160 PRESET (A,B) .....One dot at coordinates specified in variable A and variable B is erased.
170 A=A+1
180 NEXT B
190 END

```



EXAMPLE 2

```

OK
>LIST
100 REM " EXERCISE 2"
110 GCOLOR (6)
120 FTYPE (1)
130 LINE (50,100) - (250,300) , B
140 A=50
150 GCOLOR (4)
160 FOR B=100 TO 300
170 PRESET (A, B) .....
180 A=A+1
190 NEXT B
200 END

```

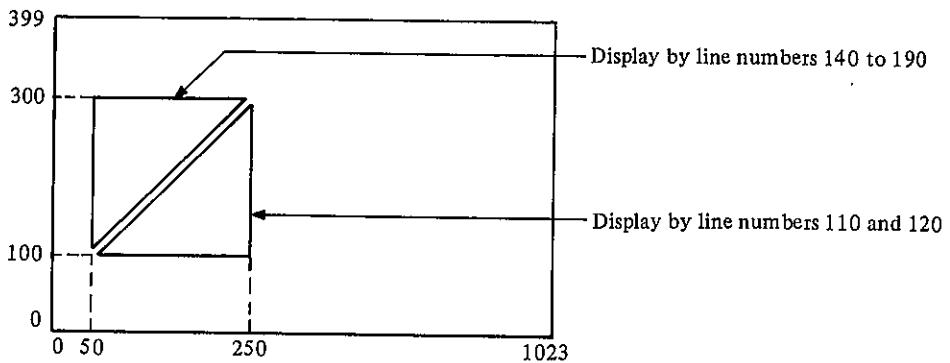
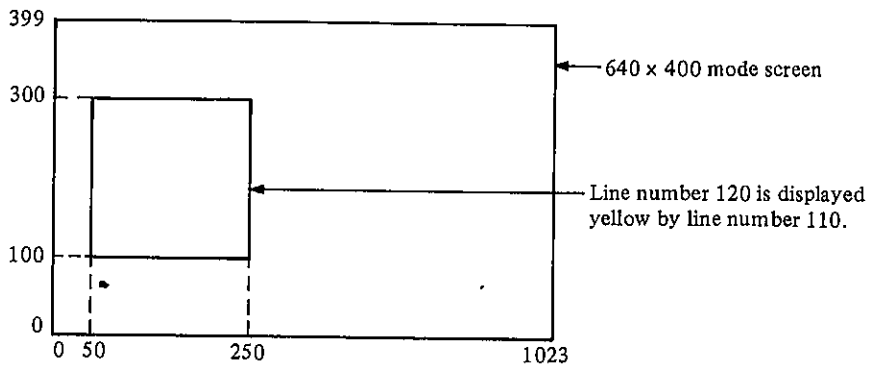
Among colors specified in line number 110, color specified in line number 150 is erased from one dot which is located at coordinates specified in variable A and variable B.

The dot is displayed red (Yellow-Green=Red)

```

OK
>RUN

```



PSET

FUNCTION

Draws one dot at desired position of screen.

FORM

PSET (X coordinate, Y coordinate)

EXPLANATION

- Draws only one dot at the position specified in [X coordinate, Y coordinate], in a color specified by the GCOLOR command (see page 189).
- When the GCOLOR command (see page 189) is omitted, one dot is drawn white.

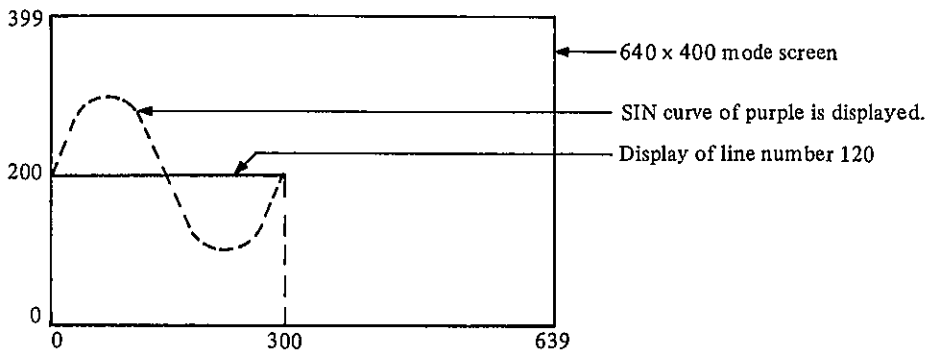
EXAMPLE

```

OK
>LIST
100 REM " EXERCISE"
110 CLS
120 LINE (0,200) - (300,200)
130 FOR A=0 TO300 STEP2
140 B=$E000
150 B(0)=100*SIN(3.14*2/300*A)+200
160 GCOLOR(3)
170 PSET(A,B(0))
180 NEXT B
190 END
OK
>RUN

```

SIN curve is drawn by one dot by line number 170 with color specified in line number 160.



VIEW

FUNCTION

Specifies the display area (view port) on the graphic screen.

FORM

VIEW (X1 position, Y1 position), (X2 position, Y2 position)

EXPLANATION

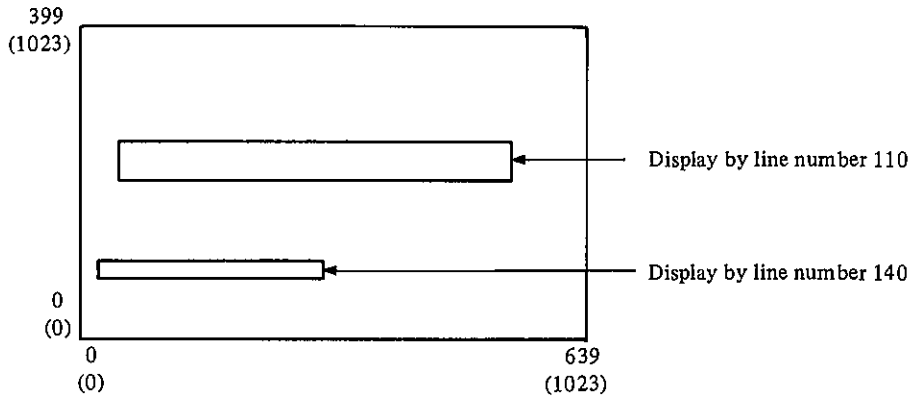
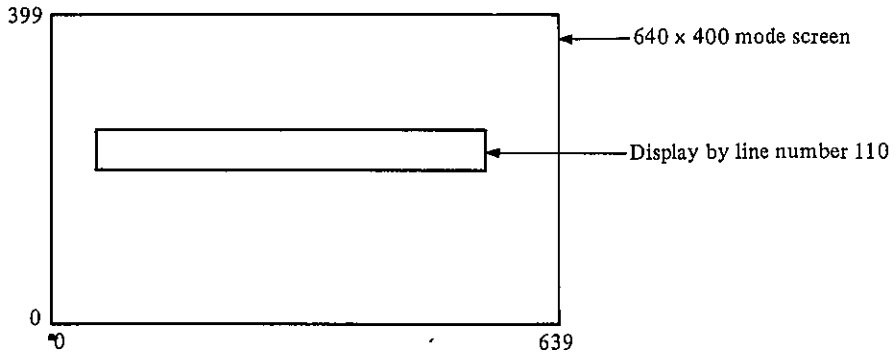
- Specifies the display area (view port) on the graphic screen.
- The display area of figure is indicated by a rectangle which has diagonal points at the point specified in [X1, Y1 positions] and the point specified in [X2, Y2 positions] by the VIEW command. Allot the coordinates of display area by use of the WINDOW command (see page 211).
- When the setting of WINDOW command is the same as the values previously set within the same program, or when the first WINDOW command in the program is set to the initial state WINDOW(0, 0), (1023, 1023)), the WINDOW command can be omitted.
- The initial state of VIEW command is a state after VIEW(0, 0), (1023, 1023) has been executed.
- By the CLS2 or CLS3 command (see page 82), only a figure drawn in the area specified by the VIEW command can be erased.
- The VIEW command only specifies the display area of figure and does not operate the actual screen. Therefore, the display position of figure, which has been drawn before the execution of VIEW command, is not moved by the VIEW command.
- The VIEW command must be set by values of "0" and larger numbers (positive numbers).

EXAMPLE 1

```

OK
>LIST
100 REM " EXERCISE 1"
110 LINE (50,200) - (550,250) , B
120 WINDOW (0,0) , (1023,1023) ... Since WINDOW command is in initial state, line number
130 VIEW (0,0) , (639,399) ..... Display area of 0 to 639 dots in X direction and 0 to
140 LINE (50,200) - (550,250) , B 39 dots in Y direction is specified, and coordinates of
150 END the area is allotted by line number 120.
OK
>RUN

```



() Numerals in parentheses are coordinates allotted by line numbers 120 and 130.

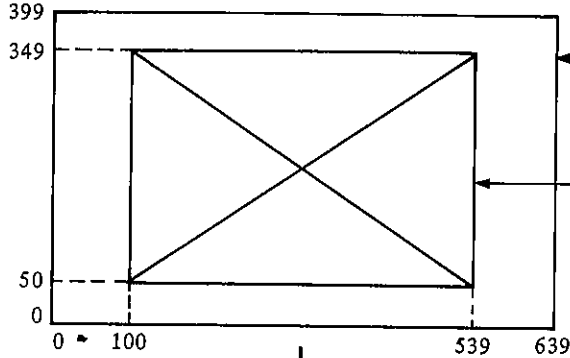
EXAMPLE 2

```

OK
>LIST
100 REM " EXERCISE 2"
110 LINE (100,50) - (539,349) , B
120 LINE (100,50) - (539,349)
130 LINE (100,349) - (539,50)
140 WINDOW (-200,-100) , (200,100)
150 VIEW (200,100) , (439,300)
160 CIRCLE (0,0) ,100
170 ZTIME500
180 CLS 2
190 END
  
```

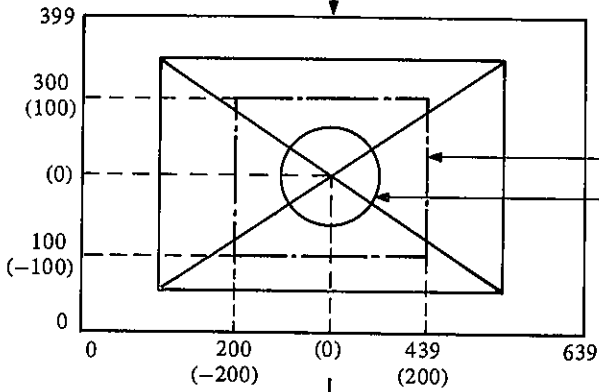
} Area of 200~439 dots in X direction and 100 to 300 dots in Y direction on the graphic screen is specified as display area, and coordinates of the area is allotted by line number 140.

OK
>RUN



← 640 x 400 mode screen

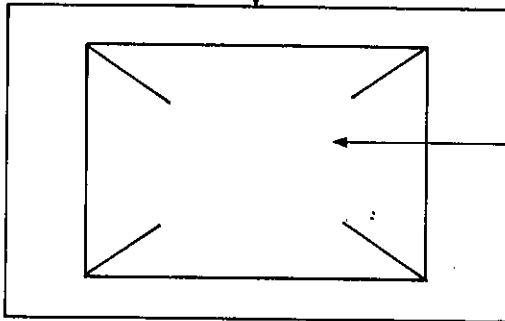
← Display by line numbers 110, 120 and 130



← Display by line number 150

← Display by line number 160

() Numerals in parentheses are coordinates allotted by line numbers 140.



← Only display area specified by VIEW command is erased by line number 180.

WINDOW

GRF

FUNCTION

Allots the coordinates of display area (view port) on the graphic display.

FORM

WINDOW (X1 coordinate, Y1 coordinate),
(X2 coordinate, Y2 coordinate)

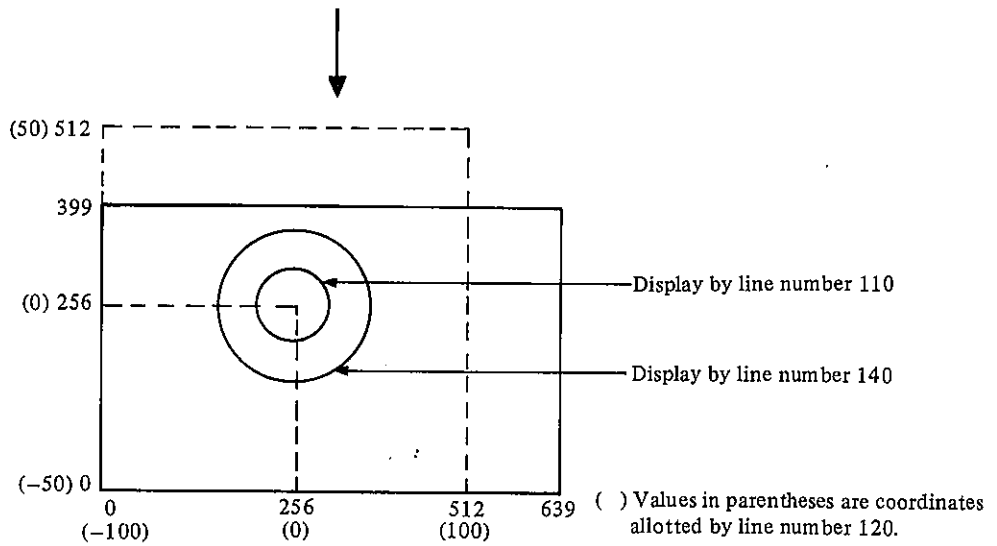
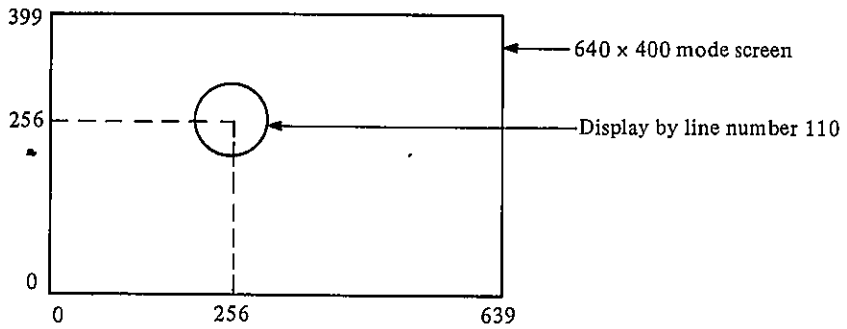
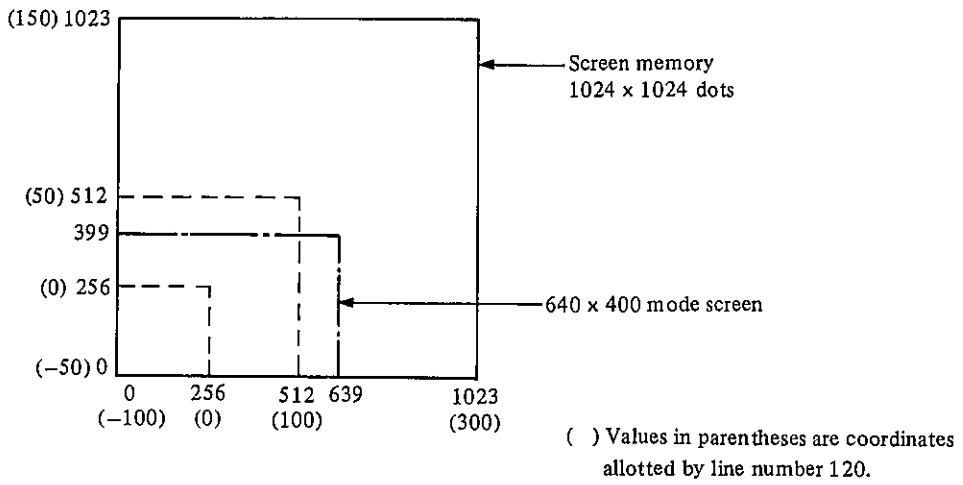
EXPLANATION

- Allots the coordinates of display area (view port) on the graphic display which has been set by the VIEW command (see page 208).
- The range of values, which can be specified in [X1, Y1, X2, Y2 coordinates], is -32767 to 32767.
- The initial state of WINDOW command is a state after WINDOW(0, 0), (1023, 1023) has been executed.
- When the setting of VIEW command is the same as the values previously set within the same program, or when the first VIEW command in the program is set to the initial state (VIEW (0, 0), (1023, 1023)), the VIEW command can be omitted.
- The WINDOW command only specifies the display area of figure and does not operate the actual screen. Therefore, the display position of figure, which has been drawn before the execution of VIEW command, is not moved by the WINDOW command.

EXAMPLE 1

```
OK
>LIST
 100 REM " EXERCISE 1"
 110 CIRCLE (256, 256) , 50
 120 WINDOW (-100, 50) , (300, 150)
 130 VIEW (0, 0) , (1023, 1023)
 140 CIRCLE (0, 0) , 40
 150 END
OK
>RUN
```

} Coordinates specified in line number 120 are allotted to area specified in line number 130. In this case, since VIEW is set to initial state, line number 130 can be omitted.

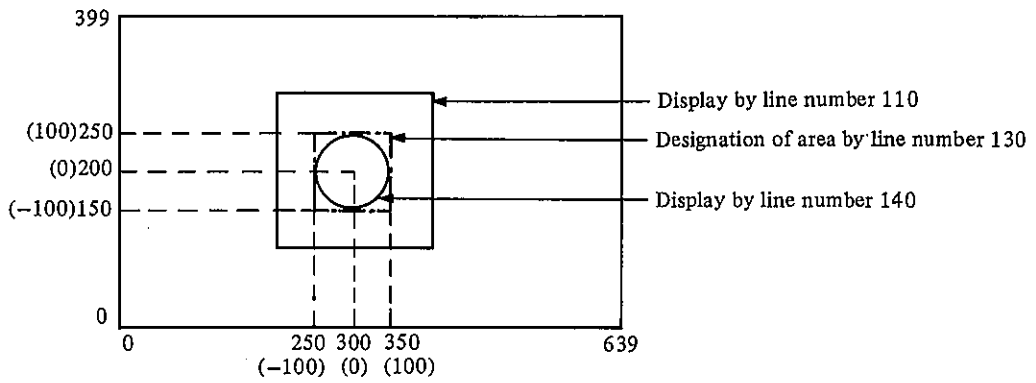
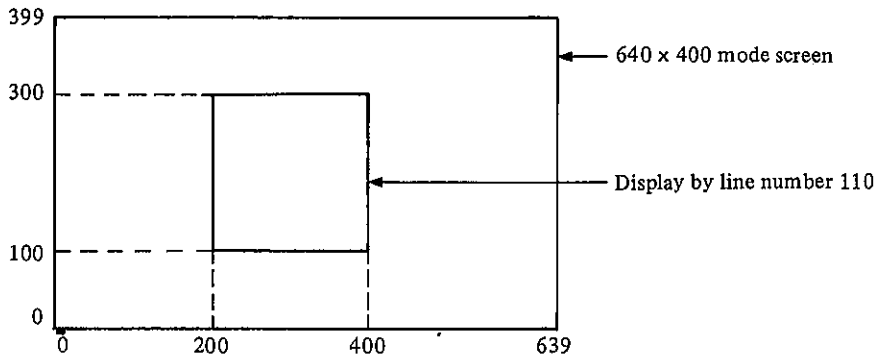


EXAMPLE 2

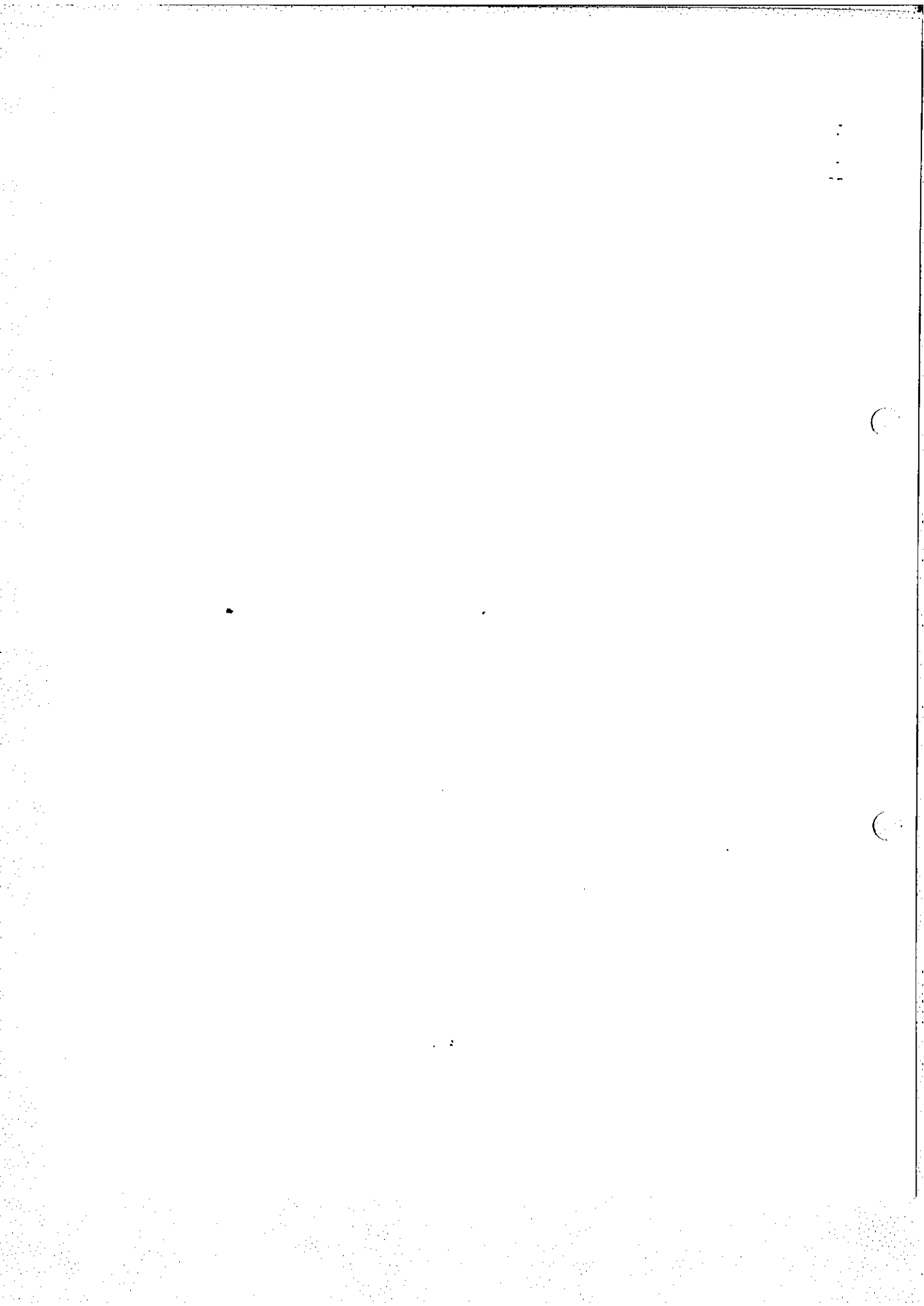
```

OK
>LIST
 100 REM " EXERCISE 2"
 110 LINE (200,100) - (400,300) ,B
 120 WINDOW (-100,-100) , (100,100)
 130 VIEW (250,150) , (350,250)
 140 CIRCLE (0,0) ,100
 150 END
OK
>RUN
  
```

Coordinates of area specified in line number 130 are specified in line number 120.



() Values in parentheses are coordinates allotted by line number 120.



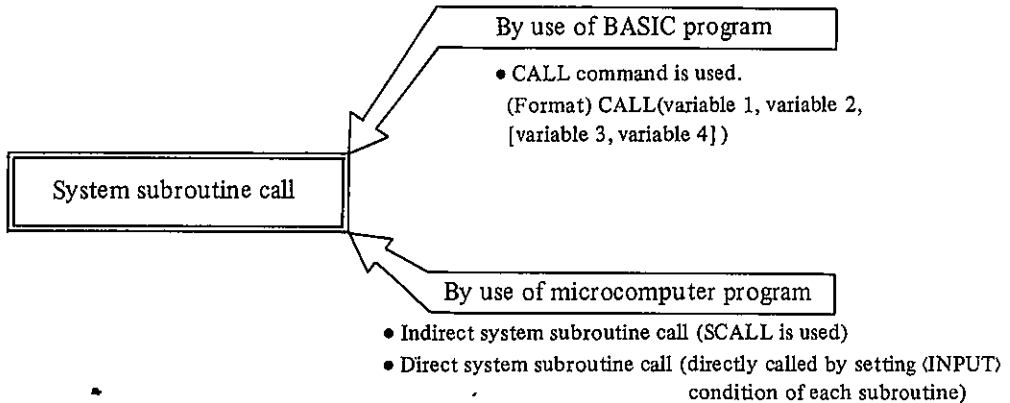
A decorative rectangular border with a repeating floral or scrollwork pattern surrounds the text.

SYSTEM SUBROUTINES

4. SYSTEM SUBROUTINES

4.1 What is system subroutine?

Each system has its subroutines unique to the system (system subroutines). The system subroutine can be used by calling it from the BASIC program by the CALL command or from the microcomputer program by the CALL instruction or indirect subroutine call.



System Subroutine Which Can Be Used Only for GPC-BASIC and Microcomputer Program					System Subroutine Which Can Be Used Only for Microcomputer Program	
SAI	SBD6	BCR	SRI	SRF	SCALL	SRSFST
SIA	SDB6	BWK	SWI	SHX	SALU	SRSFRD
SAN	SBA	SDE	SKC	SHD	SMOV	SRSFWR
SNA	SBS	SLD	SKR	SAE	SRD1	
SAF	SBM	SLE	SKP	SEA	SRD2	
SFA	SBW	SOK	SKI	SBU	SWR1	
SBF	SCA	SST	SOPEN	SBE	SWR2	
SFB	SCB	SPC	SRB	SITX	STIME	
SBD4	SMOVK	SRK	SWB	SITY	SQUIT	
SDB4	BBO	SWK	SRC	SITA		
				SITB		
				SEN	SRSFCO	

4.2 Usage of System Subroutine

The system subroutines have been prepared by the microcomputer instructions. The system subroutine can be used by calling it from the BASIC program by the CALL command or from the microcomputer program by the CALL instruction or indirect subroutine call.

4.2.1 Usage of system subroutine in BASIC program

In the BASIC program, the system subroutine is called by the CALL command.
The form of CALL command is as follows:

CALL [variable 1, variable 2, (variable 3, variable 4)]

	Application	Remarks
Variable 1	Channel of system subroutine	Depends on model. See Section 5.4.
Variable 2	Address of system subroutine	
Variable 3	Data set to (D) and (E) registers	See each system subroutine.
Variable 4	Data set to (B) and (C) registers	

CAUTION

1. It is not required to set a variable which does not have the data used for the system subroutine. (Variable 3, variable 4)
2. When the work area is used because of many variables, set the variables to the memory by use of the POKE command before executing the CALL command.
3. The CALL command is a function, and its function value is a value stored in the (H) and (L) registers when a return from the system subroutine is made.

EXAMPLE

Example of subroutine usage in BASIC program for displaying a circle (BCR)


(1) Assume that the set data are as follows:

- BCRW5080H
- Channel of system subroutine.Channel F
- Address of system subroutine.8000H
- Circle codeC3 (CC85H)
- Color codeGreen (4)

(2) Program example

```

OK
>LIST
100 POKE $5080,0.....Indicates normal display.
110 A=CALL ($F,$8000,$CC85,4)
120 IF A#0 GOTO 200
130 ].....Normal completion when A=0.
    ].....Processing at normal completion
200 ].....Processing at error
    
```

 Call of system subroutine BCR

(3) Screen display

When the above program is run, the CRT displays a 3 x 3 green circle which has its center at the current position of cursor.

When the program is normally completed, A = 0. When error exists, A = 1 or 2.

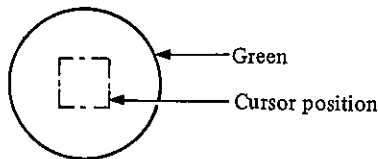


Fig. 4.1 Screen Display

4.2.2 Usage of system subroutine in microcomputer program

There are the following two types of system subroutines which can be called from the microcomputer program.

- (1) System subroutine which is called indirectly by setting the channel and address like the CALL command of BASIC program.
- (2) System subroutine which is called directly.

WARNING

Since all registers are damaged when the above types of system subroutines (1) and (2) are called, PUSH the registers, which should be protected, before calling the system subroutine.

(1) Indirect system subroutine CALL

The system subroutines indicated in Section 4.3 can be used indirectly by setting the following conditions and calling the system subroutine which is referred to as "SCALL".

	Application	Remarks
(A) register	Channel of system subroutine	Depends on model. See Section 5.4.
(H), (L) registers	Address of system subroutine	
(D), (E) registers	Data used for system subroutine	See each system subroutine.
(B), (C) registers		

NOTE

1. When the work area is used because of many data, set the data to the work area in advance.
2. Error can be judged by the value stored in (H) and (L) registers when RETURN from each subroutine is made.

EXAMPLE

Example of system subroutine usage in microcomputer program for displaying a circle (BCR)

- (1) Assume that the set data are as follows:
 - BCRW5080H
 - Channel of system subroutine.Channel F
 - Address of system subroutine.8000H
 - Circle codeC3 (CC85H)
 - Color codeGreen (4)
 - Indirect subroutine call.49H

(2) Flowchart example

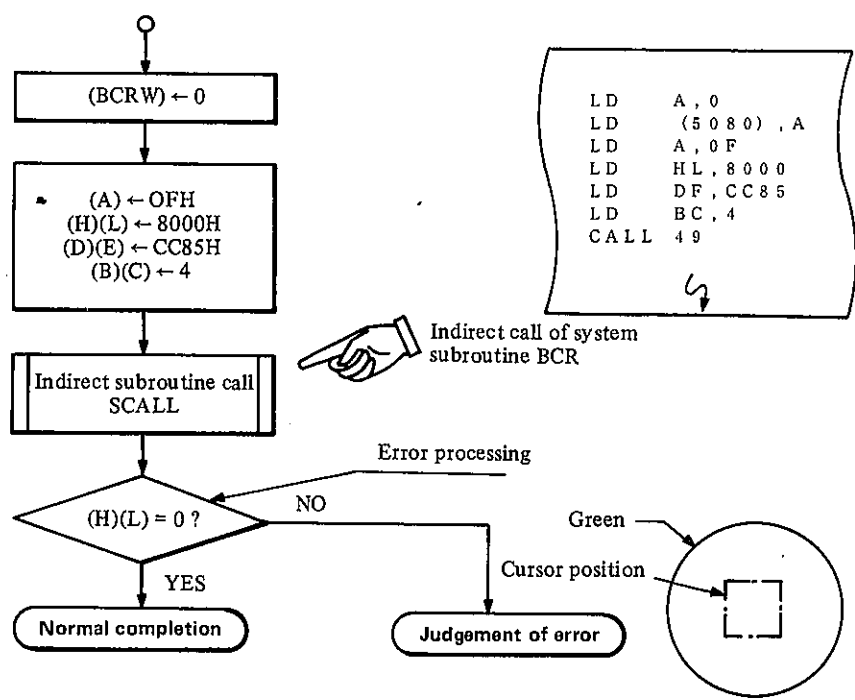


Fig. 4.2 Screen Display

(3) Screen display

When the above program is run, the CRT displays a 3 x 3 green circle which has its center at the current position of cursor.

(2) Direct subroutine CALL

The system subroutines indicated in Section 4.4 can be used by setting the INPUT conditions of each system subroutine and directly calling the system subroutine.

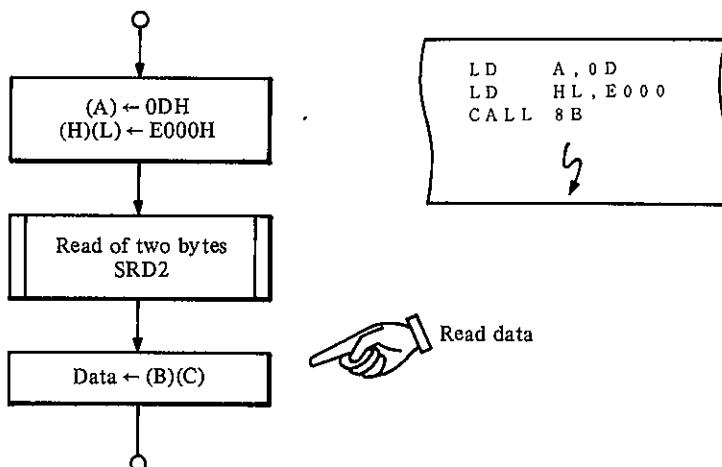
EXAMPLE

Example of system subroutine usage for reading two types from memory (SRD2)

(1) Assume that the set data are as follows:

- Channel of read data. Channel D
- Address of read data. E000H
- Address of SRD2 8BH

(2) Flowchart example



4.3 Subroutines Which Can Be Used for BASIC and Microcomputer Programs

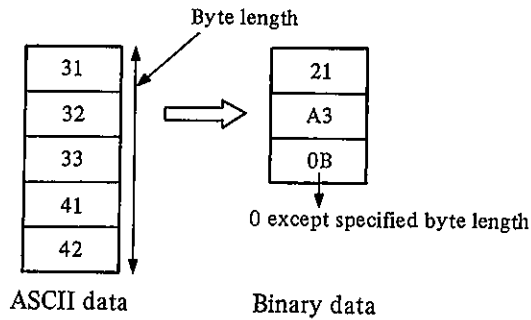
The following subroutines can be used for the BASIC and microcomputer programs.

System Subroutine	Reference Page	System Subroutine	Reference Page
SAI	223	SST	268
SIA	225	SPC	270
SAN	227	SRK	271
SNA	229	SWK	273
SAF	231	SRI	275
SFA	234	SWI	277
SBF	236	SKC	279
SFB	237	SKR	280
SBD4	239	SKP	281
SDB4	240	SKI	282
SBD6	241	SOPEN	283
SDB6	243	SRB	285
SBA	245	SWB	289
SBS	247	SRC	294
SBM	249	SRF	295
SBW	251	SHX	296
SCA	253	SHD	297
SCB	254	SAE	298
SMOVK	255	SEA	299
BBO	257	SEN	300
BCR	259	SBU	301
BWK	261	SBE	302
SDE	263	SITX	303
SLD	264	SITY	304
SLE	266	SITA	305
SOK	267	SITB	307

SAI

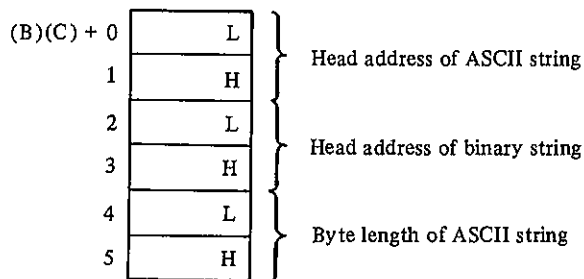
FUNCTION

Converts the specified ASCII string (hexadecimal number) into binary string.



INPUT

- (D), (E) registers Set the channel which stores the set data.
- (B), (C) registers Set the head address of set data.
- Set data. Set the head address of ASCII string, the head address of binary string, and the byte length of ASCII string to (B)(C) + 0 to 5.



OUTPUT

The execution result of SAI is stored in the (H) and (L) registers.

(H), (L) registers	Judgement
0	Specified ASCII string has been converted into binary string, and storage of conversion result in addresses (B)(C) + 2, 3 has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Set data error (Setting of ASCII string byte length is 0.)
5	Set data error (There is a code other than 0 to F in ASCII string.)

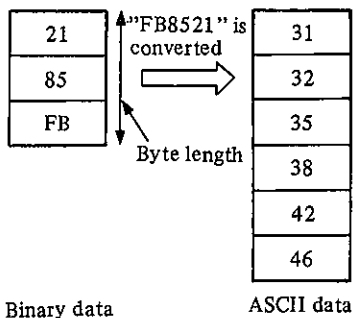
NOTE

1. Set the ASCII string to the channel specified by the (D) and (E) registers.
2. The converted binary string is stored in the channel specified by the (D) and (E) registers.

SIA

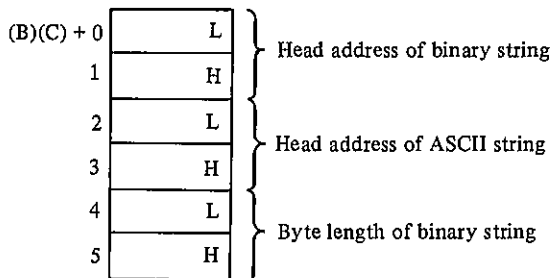
FUNCTION

Converts the specified binary string into ASCII string (hexadecimal number).



INPUT

- (D), (E) registers Set the channel which stores the set data.
- (B), (C) registers Set the head address of set data.
- Set data Set the head address of binary string, the head address of ASCII string, and the byte length of binary string to (B)(C) + 0 to 5.



OUTPUT

The execution result of SIA is stored in the (H) and (L) registers.

(H), (L) registers	Judgement
0	Specified binary string has been converted into ASCII string, and storage of conversion result in addresses (B)(C) + 2, 3 has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Set data error (Setting of ASCII string byte length is 0.)

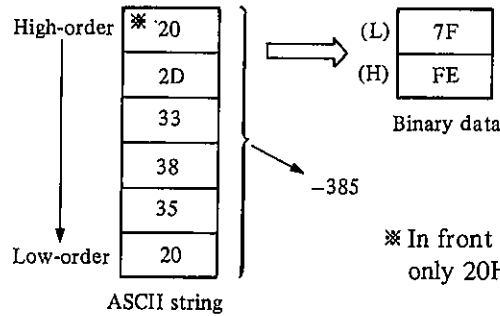
NOTE

1. Set the binary string to the channel specified by the (D) and (E) registers.
2. The converted ASCII string is stored in the channel specified by the (D) and (E) registers.

SAN

FUNCTION

Converts the ASCII string (-32767 to 32767) into binary data.



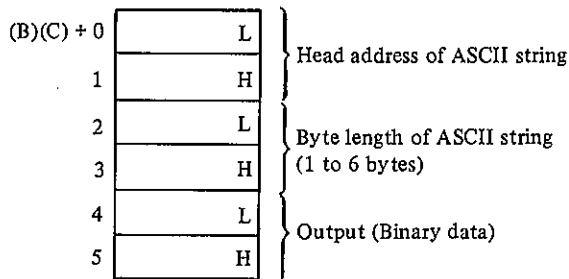
※ In front of the numerical value and sign, only 20H (space code) can be set.

INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

Set data Set the head address and byte length of ASCII string to (B)(C) + 0 to 3.



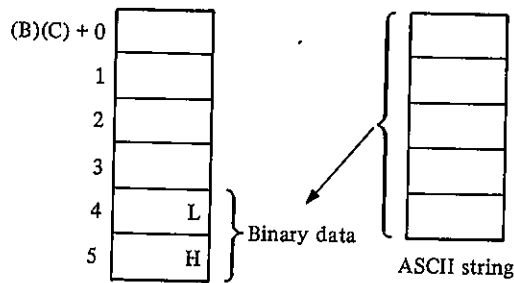
NOTE

※ Data in front of the sign and numeric value can be set to only 20#.

OUTPUT

The execution result of SAN is stored in the (H) and (L) registers.

(H), (L) registers	Judgement
0	Specified ASCII string has been converted into binary data, and storage of conversion result in addresses (B)(C) + 4, 5 has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Set data error (Byte length of ASCII string exceeds 6.)
5	ASCII data overflow (ASCII string is not -32767 to 32767.)



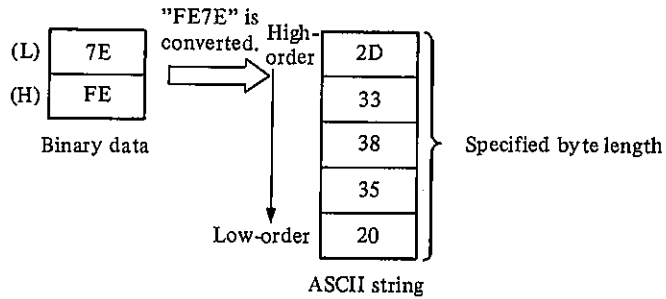
NOTE

1. Set the ASCII string to the channel specified by the (D) and (E) registers.
2. The conversion cannot be made when the ASCII string is not -32767 to 32767.
3. A negative number can also be converted into binary data.

SNA

FUNCTION

Converts the binary data into ASCII string (-32767 to 32767).

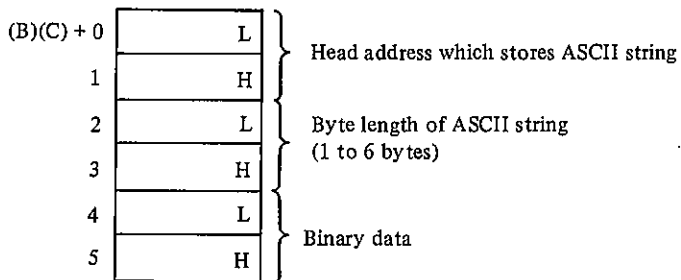


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

Set data. Set the head address which stores the ASCII string, byte length and binary data to (B)(C) + 0 to 5.



OUTPUT

The execution result of SNA is stored in the (H) and (L) registers.

(H), (L) registers	Judgement
0	Specified binary data has been converted into ASCII string, and storage of conversion result in addresses (B)(C) + 0, 1 has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Set data error (Byte length of ASCII string is not 1 to 6.)
5	ASCII data overflow (Byte length of converted ASCII string is longer than that of set ASCII string.)

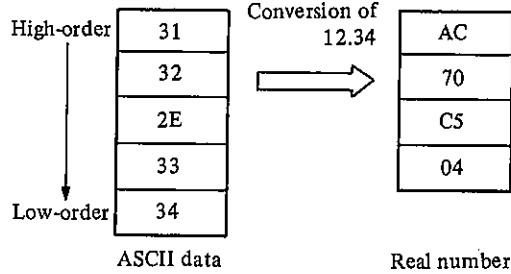
NOTE

1. The ASCII string is stored in the address of channel specified by the (D) and (E) registers.
2. A negative number can also be converted into ASCII string.

SAF

FUNCTION

Converts the specified ASCII string into a real number.

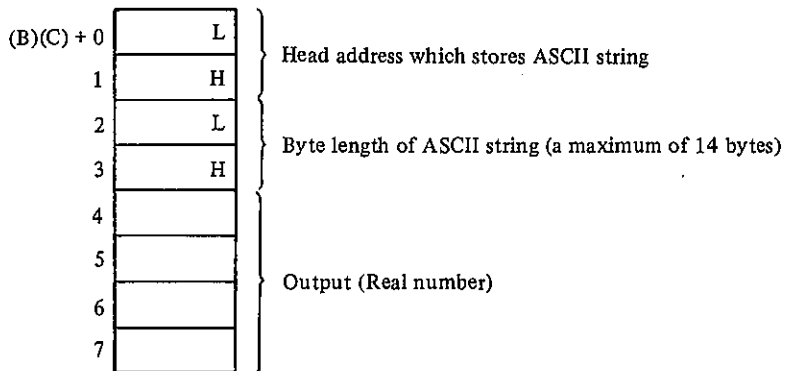


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

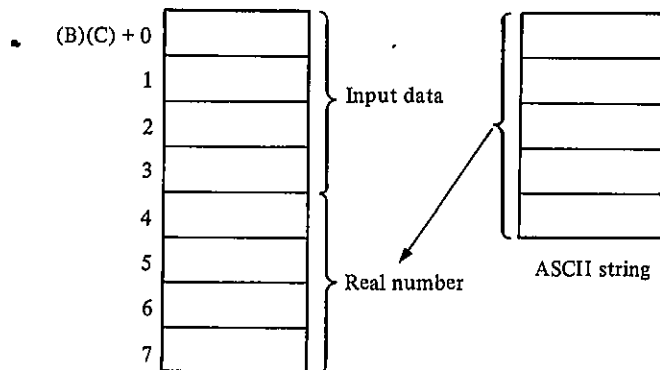
Set data. Set the head address which stores the ASCII string and byte length to (B)(C) + 0 to 3.



OUTPUT

The execution result of SAF is stored in the (H) and (L) registers.

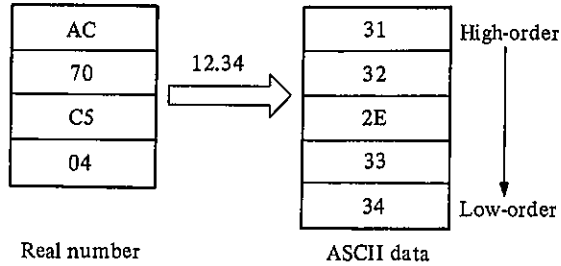
(H), (L) registers	Judgement
0	Specified ASCII string has been converted into real number, and storage of conversion result in addresses (B)(C) + 4 and thereafter has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
3	Arithmetic logic unit hardware error
4	Set data error (Byte length of ASCII string exceeds 14.)
5	Set data error (ASCII string contents error)



SFA

FUNCTION

Converts the specified real number into an ASCII string.

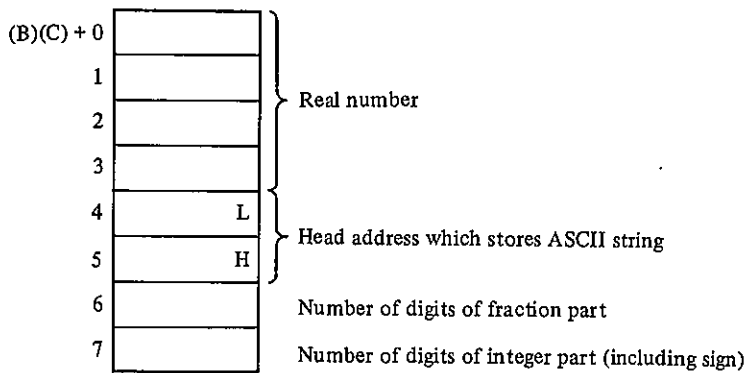


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

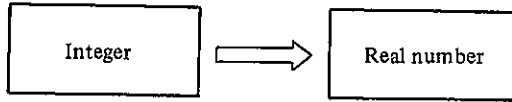
Set data. Set the real number data, head address of ASCII string, and the numbers of digits in fraction part and integer part to (B)(C) + 0 to 7.



SBF

FUNCTION

Converts an integer (-32767 to 32767) into a real number.

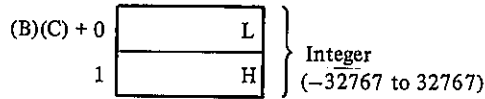


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

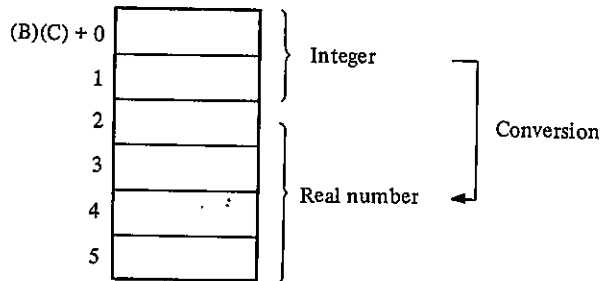
Set data. Set the integer to (B)(C) + 0 and 1.



OUTPUT

The execution result of SBF is stored in the (H) and (L) registers.

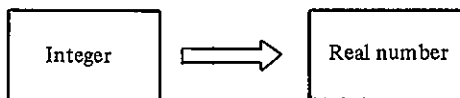
(H), (L) registers	Judgement
0	Specified integer has been converted into real number, and storage of conversion result in addresses (B)(C) + 2 to 5 has been normally completed.
1	Channel error [(D)(E) register value cannot be used for the channel.]
3	Arithmetic logic unit hardware error



SFB

FUNCTION

Converts a real number into an integer (−32767 to 32767).

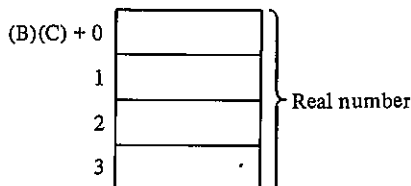


INPUT

(D), (E) registersSet the channel which stores the set data.

(B), (C) registersSet the head address of set data.

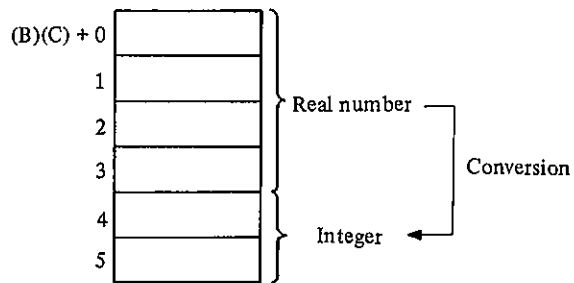
Set dataSet the real number to (B)(C) + 0 to 3.



OUTPUT

The execution result of SFB is stored in the (H) and (L) registers.

(H), (L) registers	Judgement
0	Specified real number has been converted into integer, and storage of conversion result in addresses (B)(C) + 4, 5 has been normally completed.
1	Channel error [(D), (E) register value cannot be used for the channel.]
3	Arithmetic logic unit hardware error
5	Overflow or underflow (Conversion result is not −32767 to 32767.)



NOTE

1. The digit that follows the decimal point is rounded off.

$$5.3 \rightarrow 5$$

$$-1.5 \rightarrow -2$$

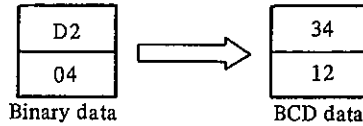
$$-1.3 \rightarrow -1$$

2. A negative number can also be converted into an integer.

SBD4

FUNCTION

Converts binary data of 0 to 9999 into BCD data.

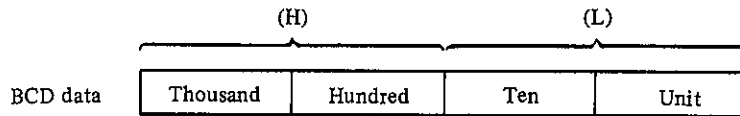


INPUT

(D), (E) registers Binary data of 0 to 9999

OUTPUT

(H), (L) registers Stores the conversion result of binary data into BCD data.



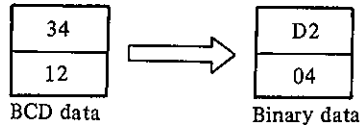
CAUTION

When a value other than 0 to 9999 is set to the (D) and (E) registers, the value is not converted correctly.

SDB4

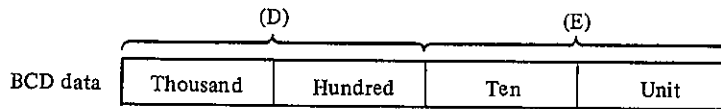
FUNCTION

Converts BCD data of 0 to 9999 into binary data.



INPUT

(D), (E) registers BCD data of 0 to 9999



OUTPUT

(H), (L) registers Stores the conversion result of BCD data into binary data. .

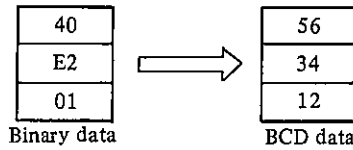
CAUTION

When each digit of (D) and (E) registers is not BCD data of 0 to 9999, the data is not converted correctly.

SBD6

FUNCTION

Converts binary data of 0 to 999999 into BCD data.

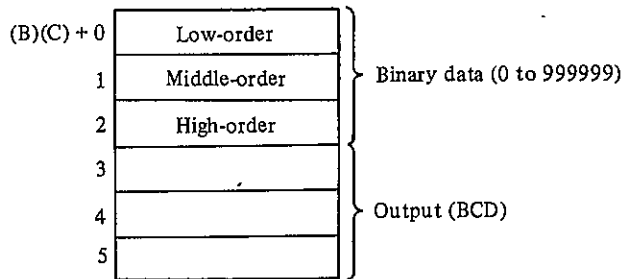


INPUT

(D), (E) registersSet the channel which stores the set data.

(B), (C) registersSet the head address of set data.

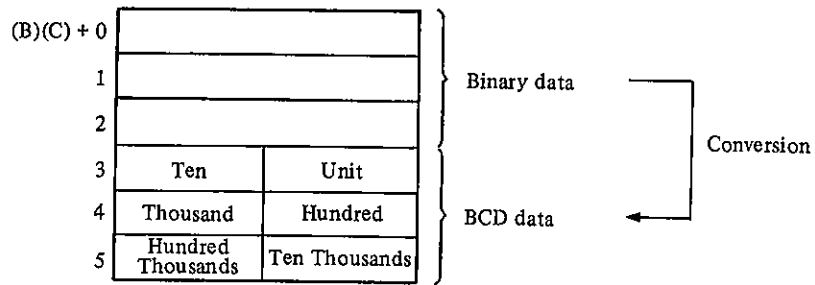
Set data.Set binary data of 0 to 999999 to (B)(C) + 0 to 2.



OUTPUT

The execution result of SBD6 is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Binary data of 0 to 999999 has been converted into BCD data, and storage of conversion result to addresses (B)(C) + 3 to 5 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]



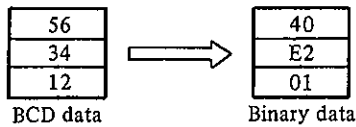
CAUTION

When the set data is a value other than 0 to 999999, the data is not converted correctly.

SDB6

FUNCTION

Converts BCD data of 0 to 999999 into binary data.



INPUT

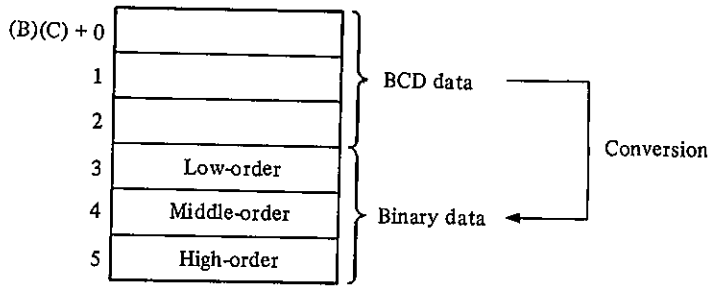
- (D), (E) registersSet the channel which stores the set data.
- (B), (C) registersSet the head address of set data.
- Set data.Set BCD data of 0 to 999999 to (B)(C) + 0 to 2.

(B)(C) + 0	Ten	Unit	}	BCD data (0 to 999999)
1	Thousand	Hundred		
2	Hundred Thousands	Ten Thousands		
3			}	Output (Binary)
4				
5				

OUTPUT

The execution result of SDB6 is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	BCD data of 0 to 999999 has been converted into binary data, and storage of conversion result to addresses (B)(C) + 3 to 5 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]



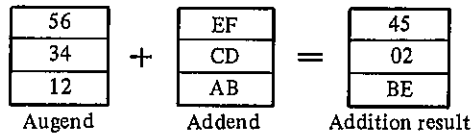
CAUTION

When the set data is a value other than BCD data of 0 to 999999, the data is not converted correctly.

SBA

FUNCTION

Performs addition of 24-bit binary data.

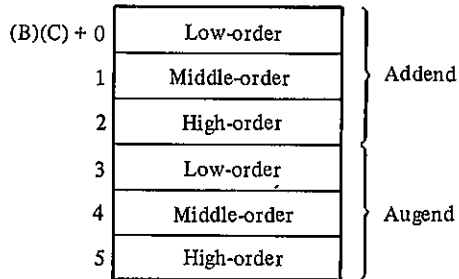


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

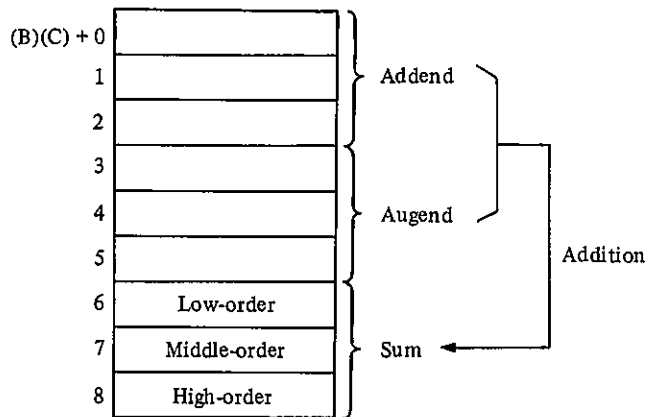
Set data Set 24-bit addend and augend to (B)(C) + 0 to 5.



OUTPUT

The execution result of SBA is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Addition of 24-bit binary data has been performed, and storage of addition result to addresses (B)(C) + 6 to 8 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]



NOTE

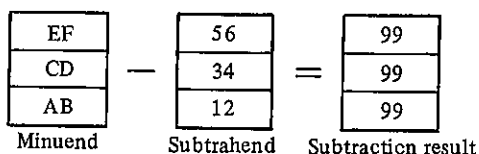
Carry is ignored.

Augend	A	=	8	0	2	A	4	5	H
Addend	B	=	B	1	4	8	C	D	H
Addition result	C	=	3	1	7	3	1	2	H

SBS

FUNCTION

Performs subtraction of 24-bit binary data.

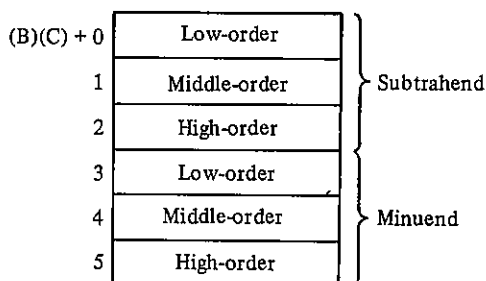


INPUT

(D), (E) registersSet the channel which stores the set data.

(B), (C) registersSet the head address of set data.

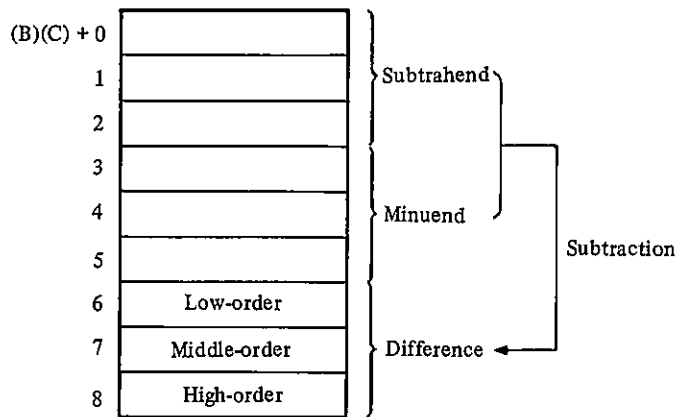
Set data.Set 24-bit subtrahend and minuend to (B)(C) + 0 to 5.



OUTPUT

The execution result of SBS is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Subtraction of 24-bit binary data has been performed, and storage of subtraction result to addresses (B)(C) + 6 to 8 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]



NOTE

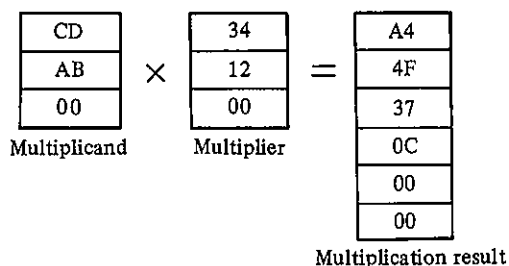
Borrow is ignored.

Minuend	A	=	1	A	5	7	B	5	H
Subtrahend	B	=	2	8	1	A	2	2	H
Subtraction result	C	=	F	2	3	D	9	3	H

SBM

FUNCTION

Performs multiplication of 24-bit binary data.

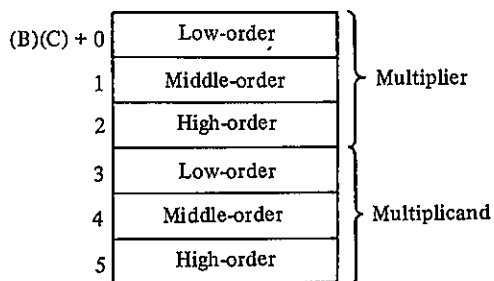


INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

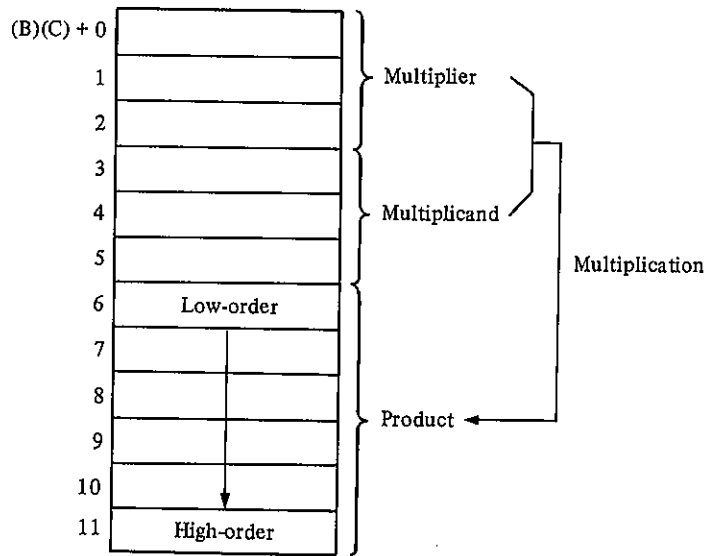
Set data Set multiplier and multiplicand to (B)(C) + 0 to 5.



OUTPUT

The execution result of SBM is stored in (H) and (L) registers.

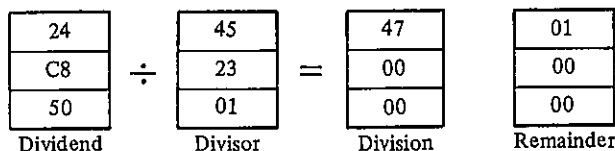
(H), (L) registers	Judgement
0	Multiplication of 24-bit binary data has been performed, and storage of multiplication result to addresses (B)(C) + 6 to 11 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]



SBW

FUNCTION

Performs division of 24-bit binary data.

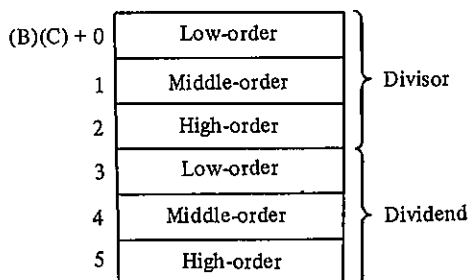


INPUT

(D), (E) registersSet the channel which stores the set data.

(B), (C) registersSet the head address of set data.

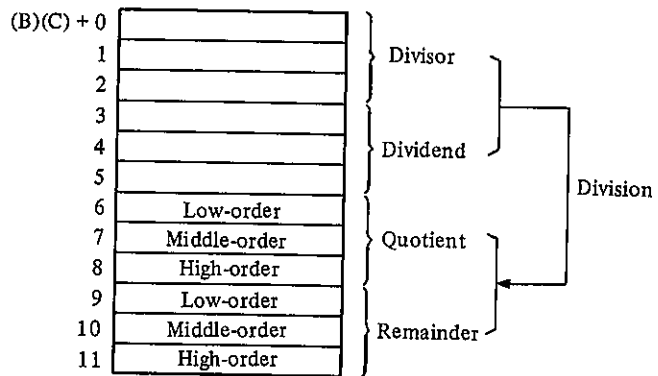
Set dataSet divisor and dividend to (B)(C) + 0 to 5.



OUTPUT

The execution result of SBW is stored in (H) and (L) registers.

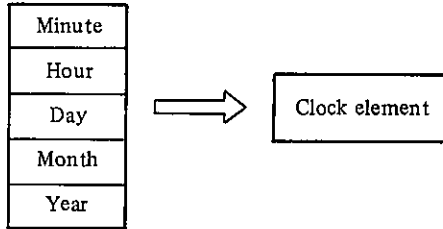
(H), (L) registers	Judgement
0	Division of 24-bit data has been performed, and storage of division result to (B)(C) + 6 to 11 has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Data error (Divisor is "0".)



SCA

FUNCTION

Writes the set data of year, month, day, hour and minute to the clock element.



INPUT

(D), (E) registers Set the channel which stores the set data.

(B), (C) registers Set the head address of set data.

Set data. Set year, month, day, hour and minute to (B)(C) + 0 to 4.

(B)(C) + 0	Minute
1	Hour
2	Day
3	Month
4	Year

OUTPUT

The execution result of SCA is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Write to clock element has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Data error (Set data has error.)

CAUTION
Specify all the set data in binary notation.

SCB

FUNCTION

Reads the present time from the clock element and stores it in the memory address specified in the (B) and (C) registers and the following addresses.

INPUT

(D), (E) registers Channel which stores the time read from the clock element.

(B), (C) registers Head address of memory which stores the time read from the clock element.

OUTPUT

The execution result of SCB is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Storage of time read from clock element in the address specified in (B)(C) registers and following addresses has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Overtime (Clock element is not placed in read state even after a predetermined time has elapsed.)

(B)(C) + 0	Minute
1	Hour
2	Day
3	Month
4	Year

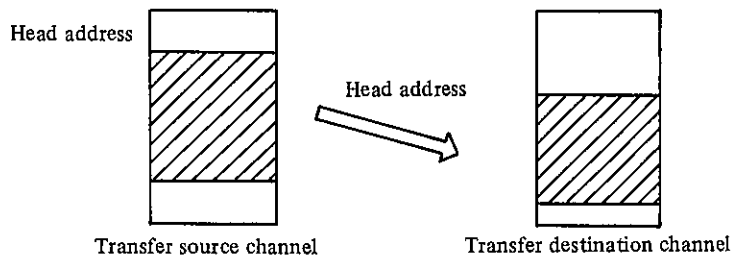
CAUTION

All the data of read time are stored in binary notation.

SMOVK

FUNCTION

Transfers data from memory to memory in block.

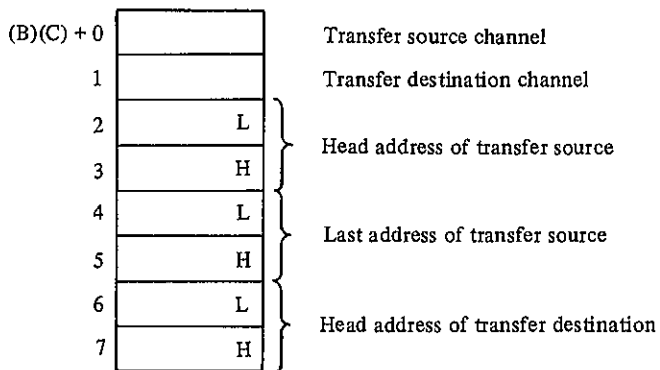


INPUT

(D), (E) registersSet the channel which stores the set data.

(B), (C) registersSet the head address of set data.

Set data.Set the transfer source channel, transfer destination channel, head address of transfer source, last address of transfer source, and head address of transfer destination to (B)(C).+ 0 to 7.



OUTPUT

The execution result of SMOVK is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Block transfer from transfer source memory to transfer destination memory has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Communication error

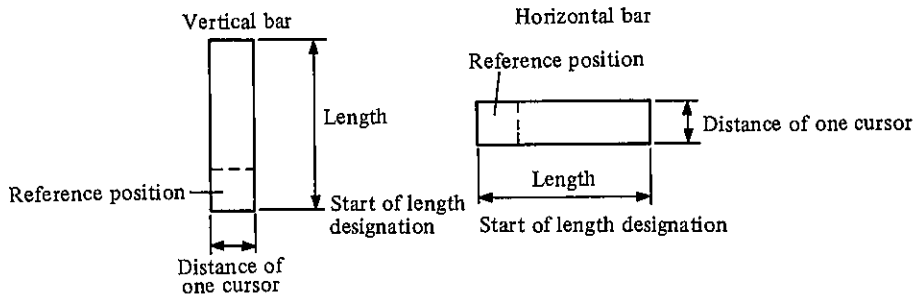
CAUTION

1. When the channel of K3NCPU has been selected, the transfer of 256 bytes interrupts the transfer for 40 msec.
2. Communication error results only when response is not given by the K3NCPU (due to hardware factor).

BBO

FUNCTION

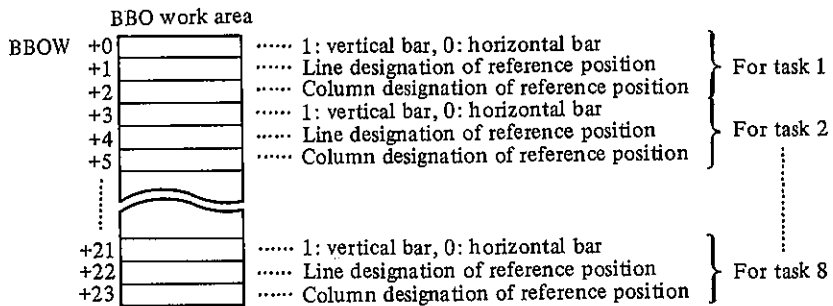
- Displays bar which have colors specified by the color codes.



- The size of cursor can be changed by the ZDSP command (see P.153).
- Set the selection of a vertical bar or horizontal bar and the reference position by task numbers to the BBO work area (BBOW).

INPUT

- (D), (E) Color code (0 to 7)
 (B), (C) Length of bar (specified by dot)



To set BBOW, write the data by use of the ZWR1 (see P.163) or SWR1 (see P.315) command.

OUTPUT

The execution result of BBO is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	Display mode is not graphic mode.





CAUTION

1. It is required to set the CRT display mode to the graphic mode by use of ZDSP command (See P.153).
2. If the BBO work area (BBOW) is set to the address for the task which does not use the BWK command, it is ignored.
3. During programming, only the BBO work area for used task is effective.
Task 2 BBOW + 3 to 5
Task 4 BBOW + 9 to 11
4. One character of standard graphic mode consists of 16 dots x 16 dots.

BCR

FUNCTION

- Displays the following circle in a color specified by the color code.

Symbol		Normal Display	Reverse Display
C0	Circle having diameter as long as two cursors is displayed.		
C1	Circle having diameter as long as one cursor is displayed.		
C2	Circle having diameter as long as two cursors is displayed.		
C3	Circle having diameter as long as three cursors is displayed.		
C4	Circle having diameter as long as four cursors is displayed.		
C5	Circle having diameter as long as five cursors is displayed.		
C6	Circle having diameter as long as six cursors is displayed.		

- The size of cursor can be changed by the ZDSP command (see P.153).
- Select normal or reverse display by changing the setting of BCR work area (BCRW). (Set 0 to BCRW for normal display, and 1 for reverse display.)
- Set the CRT display mode to the graphic mode.

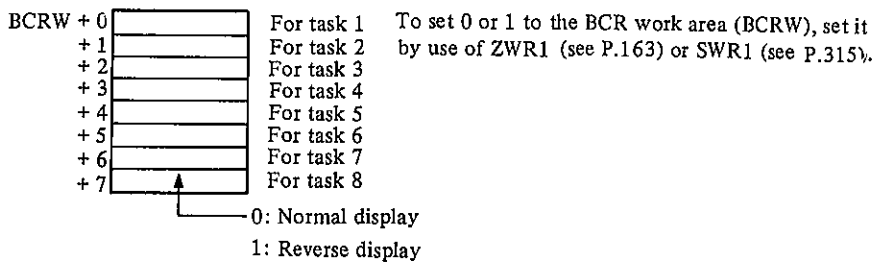
INPUT

(D), (E) registers Circle code

Symbol	Circle Code
C0	\$CBBD
C1	\$CC80
C2	\$CC81
C3	\$CC85
C4	\$CC8D
C5	\$CC99
C6	\$CCA9

(B), (C) registers Color code 0 to 7 (See the COLOR command (P.79).)

BCR work area



OUTPUT

The execution result of BCR is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	Display mode is not graphic mode.
2	Circle code setting is improper.

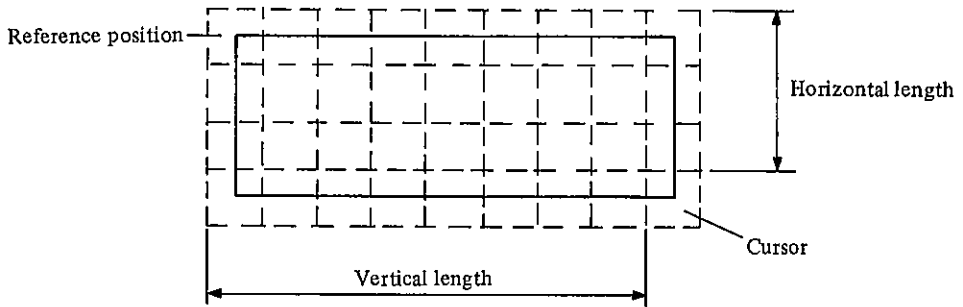
CAUTION

1. It is required to set the CRT display mode to the graphic mode by use of ZDSP command (see P.153).
2. If the BCR work area is set to the address for the task which does not use the BCR command, it is ignored.
3. During programming, only the BCR work area for used task is effective.
 - Task 2 BCRW + 1
 - Task 5 BCRW + 4

BWK

FUNCTION

- Displays the frame of color specified by the color code.



- The size of cursor can be changed by the ZDSP command (see P.153).
- Set the color code and reference position by task numbers to the BWK work area (BWKW).

INPUT

- (D), (E) registers Vertical length (Quantity of cursors)
 (B), (C) registers Horizontal length (Quantity of cursors)

BKW work area

BWKW	+ 0	Color code (0 to 7)	} For task 1
	+ 1	Line designation of reference position	
	+ 2	Column designation of reference position	} For task 2
	+ 3	Color code (0 to 7)	
	+ 4	Line designation of reference position	} For task 8
	+ 5	Column designation of reference position	

	+ 21	Color code (0 to 7)	} For task 8
	+ 22	Line designation of reference position	
	+ 23	Column designation of reference position	

To set the BWKW, write data by use of the ZWR1 (see P.163) or SWR1 (see P.315) command.

OUTPUT

The execution result of BWK is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	Display mode is not graphic mode.

CAUTION

1. It is required to set the CRT display mode to the graphic mode by use of ZDSP command (see P.153).
2. If the BWK work area (BWKW) is set to the address for the task which does not use the BWK command, it is ignored.
3. During programming, only the BWK work area for used task is effective.
Task 2BBOW + 3 to 5
Task 8BBOW + 21 to 23

SDE

FUNCTION

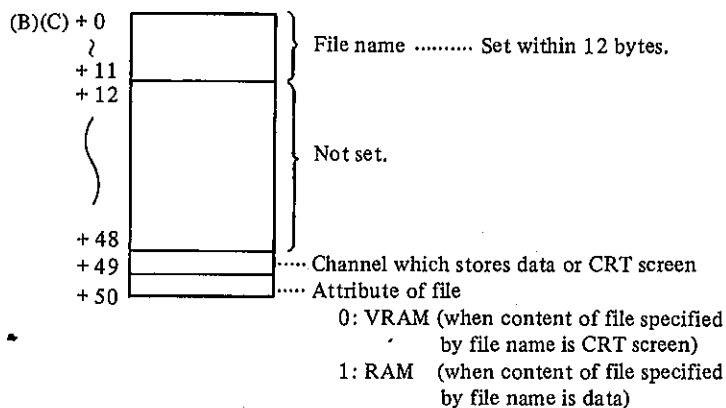
Erases desired data or all contents of CRT screen which are stored in the RAM area (32K bytes) specified for file.

INPUT

(D), (E) registersChannel which stores the set data.

(B), (C) registersHead address of set data.

Set dataSet the file name, channel which stores data or CRT screen, and the attribute of file to (B)(C) + 0 to 50.



OUTPUT

- The execution result of SDE is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	Corresponding file is not found.
2	File attribute is improper.
3	Channel setting is improper.

SLD

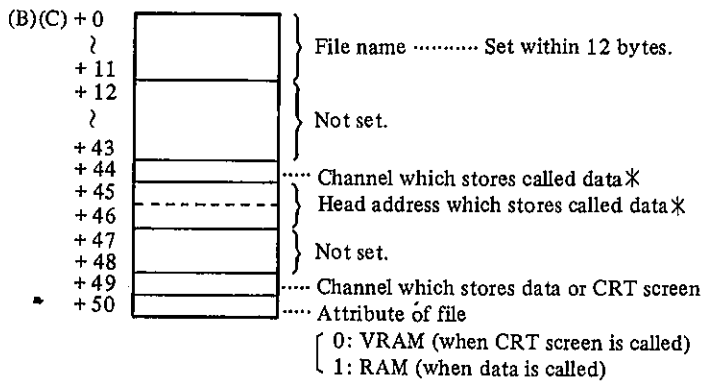
FUNCTION

Calls data or CRT screen contents from the specified file in the RAM area (32K bytes) for file.

INPUT

- (D), (E) registersChannel which stores the set data.
- (B), (C) registersHead address of set data.

Set data.Set the file name, channel which will store the called data, channel which stores the head address data or CRT screen, and the attribute of file to (B)(C) + 0 to 50.



Channel number and head number with * marks are required only when the attribute of file is 1 (RAM).

OUTPUT

- The execution result of SLD is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	The same file name exists.
2	File attribute is improper.
3	Channel setting is improper.
4	Address setting is improper.

CAUTION

When the attribute of file is VRAM, the screen mode for write by use of SST and the screen mode for read by use of SLD must be the same.

SLE

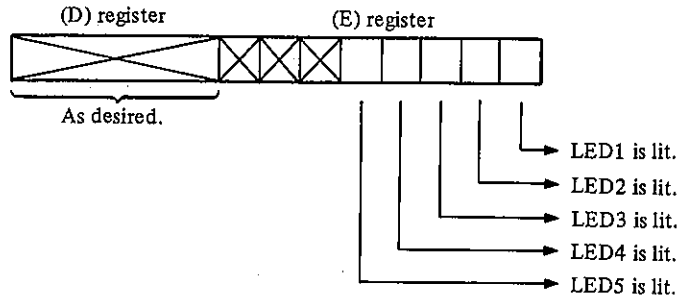
FUNCTION

Lights desired LED (LED 1 to 5) on the operation panel.

INPUT

(D), (E) registersSpecify lighting data.

The structure of lighting data expressed in binary number is as follows. Set "1" only to the LED which is desired to be lit.



SOK

FUNCTION









Judges whether or not the key of operation panel has been pressed.

When the key has been pressed, its key code (20H to 5FH) is returned to the (H) and (L) registers. When it has not been pressed, FFH is returned to the (H) and (L) registers.

The difference between the SOK command and INKEY command (see P.128) is; while the INKEY command interrupts execution until the key is pressed, the SOK command returns the state of key at the execution of command to the (H) and (L) registers, and continues execution.

OUTPUT

(H), (L) registers Hexadecimal key codes shown in the following table are returned.

Code	20	28	30	38	40	48	50	58
Description	Operation monitor	F1	F2	F3	Circuit F4	List Screen preparation	SCPU Screen copy	Setting
Code	21	29	31	39	41	49	51	59
Description	Read	Write	Conversion	Insertion	Deletion	Monitor	Verify	Test
Code	22	2A	32	3A	42	4A	52	5A
Description					MOV Index	> Preceding page	+ Upward scroll	BCD Standard
Code	23	2B	33	3B	43	4B	53	5B
Description					= Screen off	< Next page	- Downward scroll	BIN Double magnification
Code	24	2C	34	3C	44	4C	54	5C
Description	↑	X	T	LD C	AND D	OR E	MC F	NOP Quadruple magnification
Code	25	2D	35	3D	45	4D	55	5D
Description	→	Y	C	LDI 8	ANI 9	ORI A	MCR B	END Completion
Code	26	2E	36	3E	46	4E	55	5E
Description	←	M	D	SET 4	ANB 5	ORB 6	PLS 7	Clear
Code	27	2F	37	3F	47	4F	57	5F
Description	↓	F	K	RST 0	SET 1	CJ 2	OUT 3	GO

SST

FUNCTION

Stores data or CRT screen contents in the RAM area (32K bytes) specified for file.

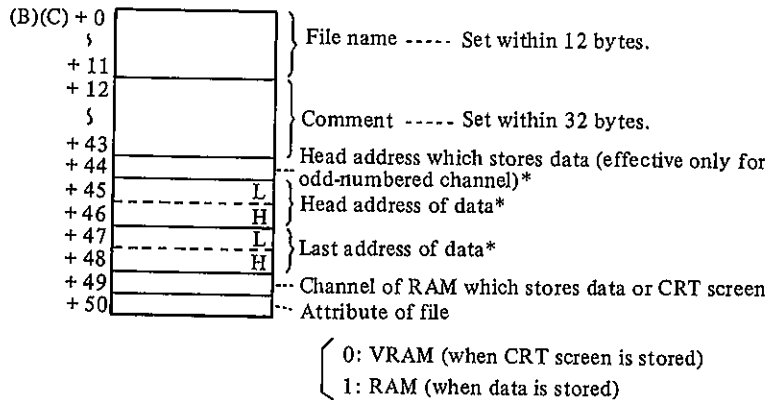
A maximum of 39 files or a file of up to 30K-byte capacity can be stored in the RAM area (32K bytes).

INPUT

(D), (E) registersChannel which stores the set data.

(B), (C) registersHead address of set data.

Set data.Set the file name, the comment, the head address of channel number which stores data, the last address of channel number which stores data, the channel of RAM which will store data or CRT screen, and the attribute of file to (B)(C) + 0 to 50.



Channel number, head address and last address with * marks are required only when the attribute of file is 1 (RAM).

OUTPUT

- The execution result of SST is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Execution has been completed normally.
1	The same file name exists.
2	File area is full.
3	Channel setting is improper.
4	Address setting is improper.

- The number of files is stored in address \$F800 of RAM (32K bytes).

CAUTION

1. Do not create a program in the RAM area which has been specified to store the data or CRT screen.
2. Before storing the data or CRT screen in the specified RAM area, write "0" to the address \$F800 of RAM area to initialize it.
3. When "0" (VRAM) has been selected as the attribute of file, the whole screen of CRT is stored in the RAM area.

SPC

FUNCTION

Discriminates the loaded programmable controller CPU.

INPUT

None

OUTPUT

The discrimination result of programmable controller CPU is stored in the (H) and (L) registers.

(H), (L) registers	Discrimination
0	K2CPU-S3, K2HCPU, K2NCPU
1	K3NCPU(P2)
2	Communication error with programmable controller CPU

SRK

FUNCTION

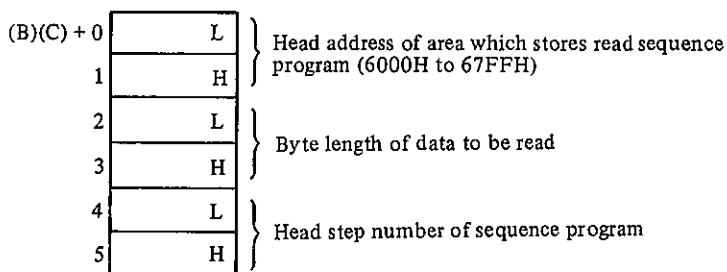
Reads one circuit of sequence program in circuit format to the specified memory.

INPUT

(D), (E) registersSet channel which stores the set data.

(B), (C) registersSet the head address of set data.

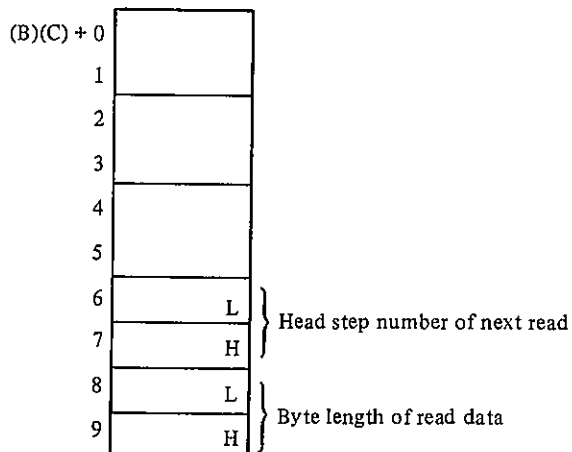
Set data.Set the head address of area which stores read sequence program, byte length of sequence program, and head step number of read sequence program to (B)(C) + 0 to 5.



OUTPUT

- The execution result of SRK is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	SRK has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Communication error with programmable controller CPU
4	Input data error
5	Circuit conversion error
6	In-use error (During read/write of sequence program by another task)
8	Circuit end (in the case of END instruction)



NOTE

1. Program can be read even if the programmable controller CPU is running.
2. Access time to programmable controller CPU can be set by the system subroutine SKI.
3. When access time to programmable controller CPU has not been set, program is read per 10 ms without regard to the run or stop of programmable controller CPU.

SWK

FUNCTION

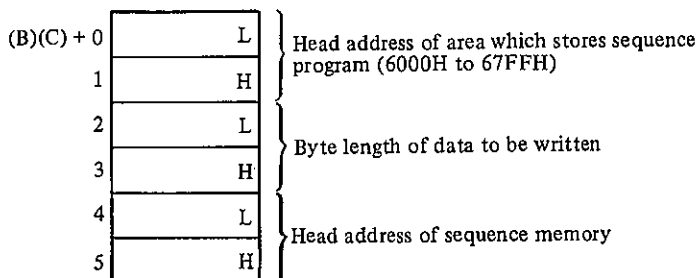
Writes one circuit of sequence program to the sequence memory.

INPUT

(D), (E) registersSet channel which stores the set data.

(B), (C) registersSet the head address of set data.

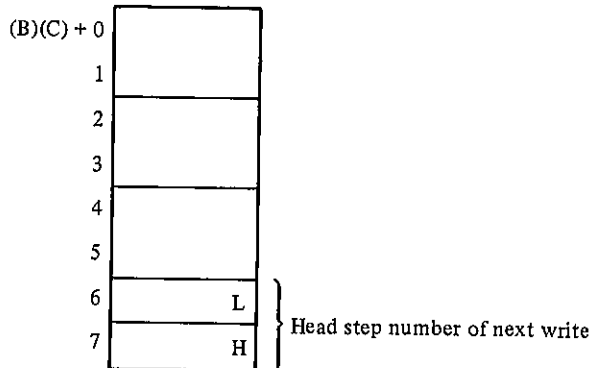
Set data.Set the head address which stores sequence program, byte length of sequence program, and head address of sequence memory to (B)(C) + 0 to 5.



OUTPUT

The execution result of SWK is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Write of sequence program to sequence memory has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Communication error with programmable controller CPU
4	Input data setting error
5	Circuit data error
6	In-use error (During read/write of sequence program by another task)
7	During run of programmable controller CPU



CAUTION

1. Program cannot be written during run of programmable controller CPU.
2. In K2HCPU or K3NCP(P2), program can be written after performing remote stop by use of the system subroutine SKP.
3. The programmable controller CPU can be always accessed. (The setting of system subroutine SKI can also be set to "0".)

SRI

FUNCTION

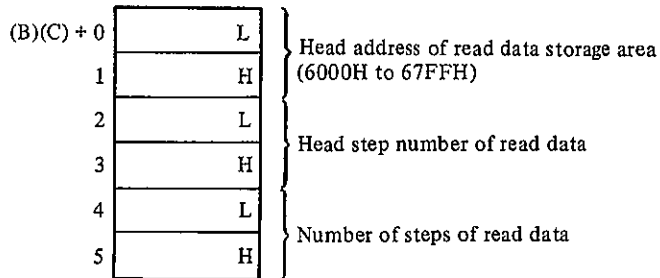
Reads specified steps of sequence program in list format to the specified memory.

INPUT

(D), (E) registersSet channel which stores the set data.

(B), (C) registersSet the head address of set data.

Set dataSet the head address of read data storage area, head number of read data, and the number of steps to (B)(C) + 0 to 5.



OUTPUT

The execution result of SRI is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Read of sequence program to specified memory has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Communication error with programmable controller CPU
4	Input data error
6	In-use error

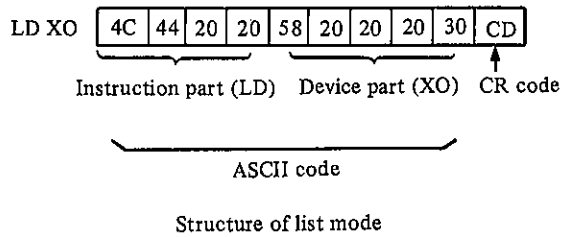
CAUTION

1. 10 bytes of read data are required per step. Secure the area which stores the read data, by the number of bytes of (the number of read steps) x 10.
2. Program can also be read during run of programmable controller CPU.
3. Interval of access time to programmable controller CPU can be set by the system subroutine SKI.
4. When interval of access time has not been set by the system subroutine SKI, sequence program is read per 10 ms from the memory of programmable controller CPU.

SWI

FUNCTION

Writes list mode data of sequence program to the sequence memory.

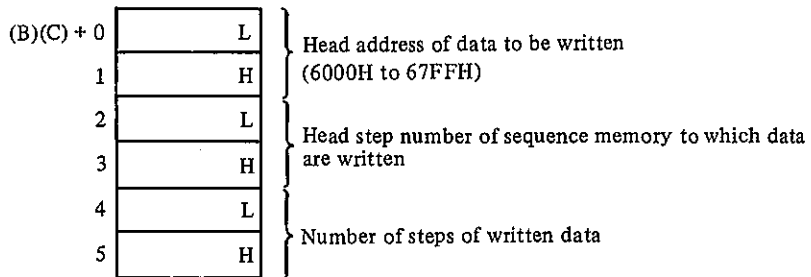


INPUT

(D), (E) registersSet channel which stores the set data.

(B), (C) registersSet the head address of set data.

Set dataSet the head address of data to be written, head step number of sequence memory to which data are written, and the number of steps to (B)(C) + 0 to 5.



OUTPUT

The execution result of SWI is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Write of sequence program to the step specified in (B)(C) + 2, 3 and following steps has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
2	Communication error with programmable controller CPU
4	Input data setting error
5	ASCII data error
6	In-use error

CAUTION

1. Program can also be written during run of programmable controller CPU. However, it is recommended, for the sake of safety, to restrict write to the changing of timer/counter set values, etc. If illogical rewriting is executed, the CPU stops and the RUN LED flickers, possibly resulting in an accident.
2. The interval of access time to programmable controller CPU can be set by the system subroutine SKI.
When the interval of access time has not been set by SKI, the sequence program program is written to the sequence memory per 10 ms without regard to the run or stop of programmable controller CPU.

SKC

FUNCTION

Checks the RUN/STOP state of loaded programmable controller CPU.

INPUT

None

OUTPUT

RUN/STOP state is stored in (H) and (L) registers.

(H), (L) registers	Judgement
0	Stop
1	Run
2	Communication error with programmable controller CPU

SKR

FUNCTION

Performs the remote run of programmable controller CPU.

INPUT

None

OUTPUT

The execution result of SKR is stored in (H) and (L) registers.

(H), (L) Registers	Judgement
0	Remote run has been completed normally.
2	Communication error with programmable controller CPU
7	Error (The programmable controller CPU remains in stop state.)

WARNING

SKR cannot be used for the K2CPU-S3.

SKP

FUNCTION

Performs the remote stop of programmable controller CPU.

INPUT

(D), (E) registers Data clear condition [can be set only to K3NCPU(P2).]

(D), (E) Registers	Condition
0	All data are cleared.
1	Only non-latched data are cleared.
2	Data are not cleared.

OUTPUT

The execution result of SKP is stored in (H) and (L) registers.

(H), (L) Registers	Judgement
0	Remote stop has been completed. K3(N)CPU(P2). . .Data clear is executed depending on input condition. K2HCPU. Data clear is not executed.
2	Communication error with programmable controller CPU
4	Input data error [0 to 2 is not in (D) and (E) registers.]
7	Error (The programmable controller CPU remains in run state.)

WARNING

1. In K2HCPU, data clear is not executed. Therefore, perform clear operation by use of command such as ZWR1 or ZWR2.
2. SKP cannot be used for the K2CPU-S3.
3. When there is a unit which performs remote stop, other than the programmable controller CPU, SKP cannot be used.

SKI

FUNCTION

Sets the interval of access time to the programmable controller CPU.

INPUT

(D), (E) registers Interval of access time to the programmable controller CPU (0 to 255)
Unit: ms

OUTPUT

The execution result of SKI is stored in (H) and (L) registers.

(H), (L) Registers	Judgement
0	Setting of interval of access time to programmable controller CPU has been completed.
4	Error

CAUTION

1. When the interval of access time has not been set by SKI, the programmable controller CPU is accessed per 10 ms.
2. Set the interval of access time by use of SKI also when the read/write of vast data is performed from/to the programmable controller CPU by use of command, such as ZMOV, SRD1, SRD2, SWR1 or SWR2, during run of the programmable controller CPU.

SOPEN

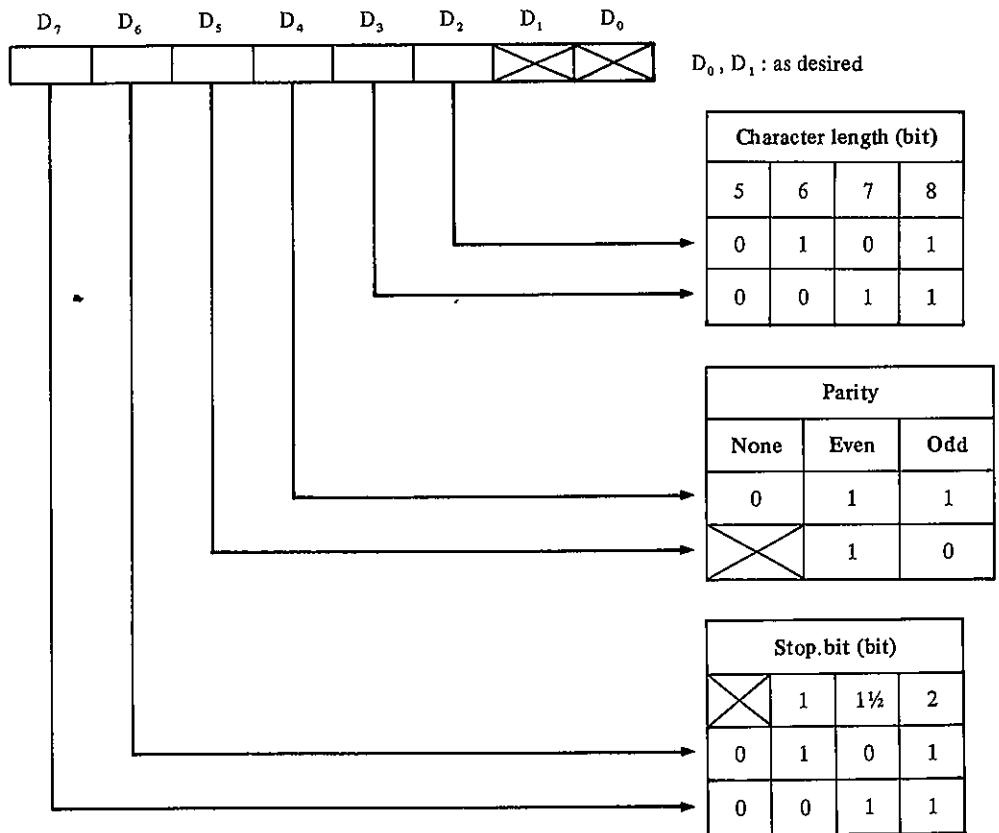
FUNCTION

Opens the channel of RS-232C and enables sending and receiving operations. This command has the same function as the OPEN command (see P.108).

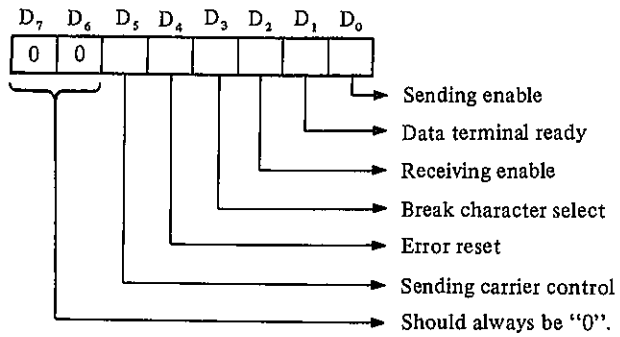
INPUT

- (B) register. Channel number of RS-232C for sending and receiving operations.
- (C) register. Set the mode as shown below.
- (D) register. Set the command as shown below.
- (E) register. Set the baud rate as shown below.

The structure of mode setting expressed in binary notation is as shown below:



The structure of command setting expressed in binary notation is as shown below. Each bit is effective when it is "1".



The set values of baud rate are as follows:

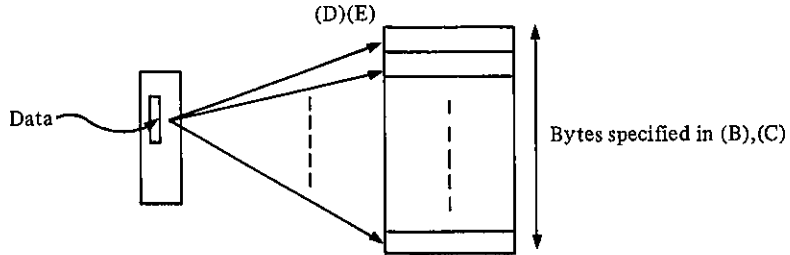
Set Value	1	2	3	4	5	6
Baud rate (Baud)	300	600	1200	2400	4800	9600

SRB

TYPE	A
------	---

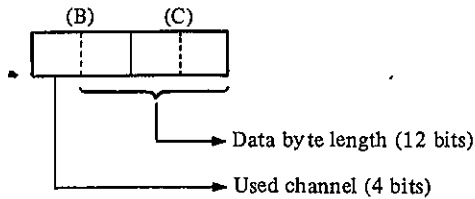
FUNCTION

- After called, this subroutine receives the specified byte length of data sent to the specified channel.
- When the receiving of specified byte length has been completed, execution returns to the program which has called SRB.



INPUT

- (D), (E) registers Set the head address of data storage area.
- (B), (C) registers Set the data byte length and used channel.

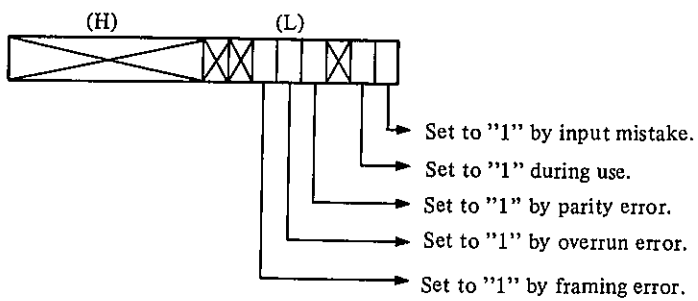


OUTPUT

After execution of SRB, the contents of (H) and (L) registers tell which error has occurred.

Carry Flag	Judgement
0	Execution of SRB has been completed.
Other than 0	Error (For error types, see the following page.)

Error Types



SRB

TYPE	B
------	---

FUNCTION

After called, this subroutine receives the specified byte length of data sent to the specified channel.

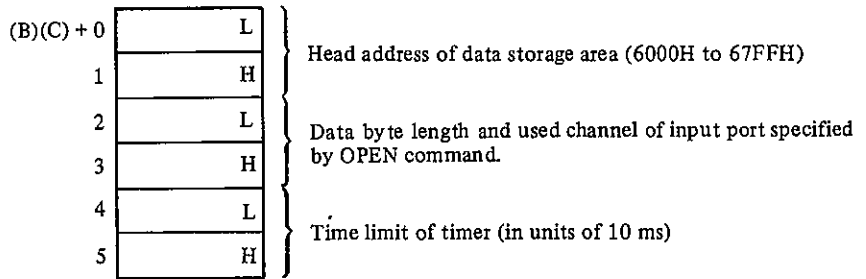
When the specified byte length cannot be received even after the receiving of specified byte length has been completed or (time limit of timer) x 10 ms has elapsed, execution returns to the program which has called SRB.

INPUT

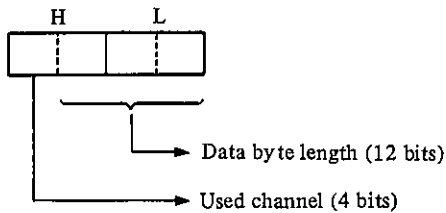
(D), (E) registers Set the head address of data storage area.

(B), (C) registers Set the address of set data.

Set data Set the head address of data storage area, data byte length, used channel, and time limit of timer to (B)(C) + 0 ~ 5.



Designation of data length and used channel

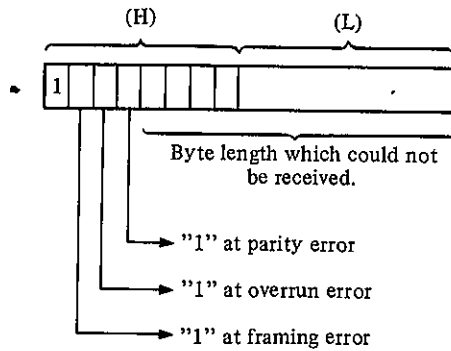


OUTPUT

The execution result of SRB is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SRB has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Data error (Data storage address, channel error)
6	SRB or SWB is being executed at specified channel.
Other than above	Overtime, parity error, overrun error, framing error

Contents of (H), (L) registers at overtime



WARNING

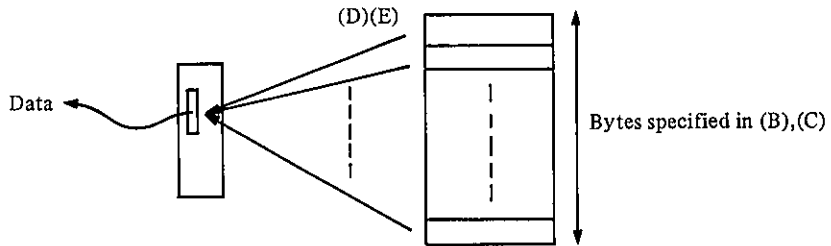
1. Do not use the same channel for plural tasks.
2. Overtime occurs when the receiving of specified byte length is not completed within (time limit of timer) x 10 ms.

SWB

TYPE	A
------	---

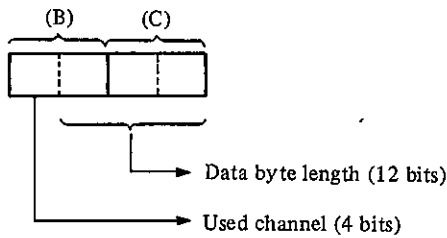
FUNCTION

- After called, this subroutine sends the specified byte length of data from the specified channel.
- When the sending of specified byte length has been completed, execution returns to the program which has called SWB.



INPUT

- (D), (E) registers . . . Set the head address of data storage area.
 (B), (C) registers . . . Set the data byte length and used channel.

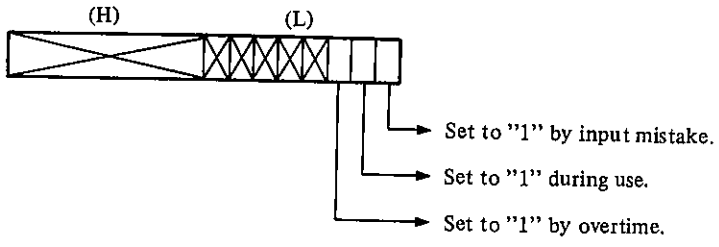


OUTPUT

- The execution result of SWB is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SWB has been completed normally.
Other than 0	Error (For error types, see the following page.)

Error Types



SWB

TYPE	B
------	---

FUNCTION

- After called, this subroutine sends the specified byte length of data from the specified channel.
- When the sending of specified byte length has been completed or when the specified byte length cannot be sent within the period of (time limit of timer) x 10 ms, execution returns to the program which has called SWB.
- When the system subroutine SHD has been called, data sending is controlled as shown below:

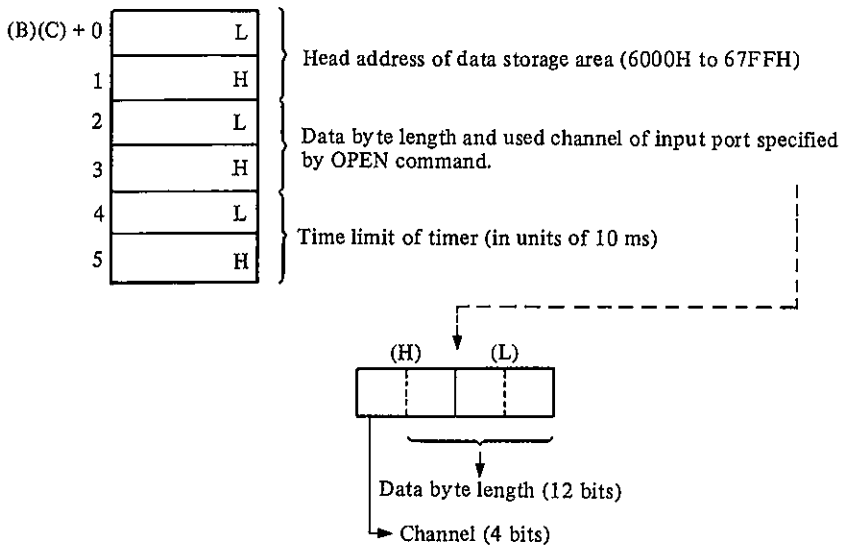
Data Set Ready	High Level	Low Level
Execution/non-execution of sending	Execution	Non-execution

- When the system subroutine SHX has been called, data sending is controlled as shown below:

Xon/Xoff Setting	Xon	Xoff
Execution/non-execution of sending	Execution	Non-execution

INPUT

- (D), (E) registers Set the head address of data storage area.
- (B), (C) registers Set the head address of set data.
- Set data Set the head address of data storage area, data byte length, channel, and time limit of timer to (B)(C) +0 to 5.

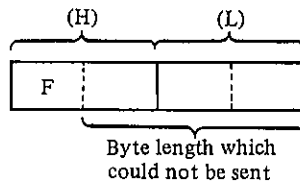


OUTPUT

The execution result of SLD is stored in (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SWB has been completed normally.
1	Channel error [(D), (E) register value cannot be used for the channel.]
4	Data error (Data storage address, channel error)
6	SWB or SRB is being executed at specified channel.
Other than above	Overtime (See the following page.)

Contents of (H), (L) registers at overtime



CAUTION

1. Overtime occurs when data cannot be set even after (time limit of timer) x 10 ms has elapsed because the send buffer is full.
2. Do not use the same channel for plural tasks.
3. The send buffer is of 128 bytes.

SRC

FUNCTION

Reads the number of data bytes which are used by the receive buffer of specified channel.

INPUT

(D), (E) registers Set the channel of RS-232C or RS-422.

OUTPUT

The execution result of SRC is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0 ~ 511	The number of bytes used by receive buffer
FFFFH	Channel error

NOTE

The receive buffer is of 511 bytes.

SRF

FUNCTION

Reads the number of bytes in the vacant area of receive buffer of specified channel.

INPUT

(D), (E) registers Set the channel of RS-232C or RS-422.

OUTPUT

The execution result of SRF is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0 ~ 511	The number of bytes in the vacant area of receive buffer
FFFFH	Channel error

NOTE

1. The receive buffer is of 511 bytes.
2. When the number of bytes in the vacant area of receive buffer is zero, data received from exterior cannot be stored in the receive buffer.

SHX

FUNCTION

Controls the data sending and receiving of RS-232C or RS-422 channel.

During sending

- When the Xoff code is received, stops data sending to exterior.
- When the Xon code is received, resumes data sending to exterior.

During receiving

- When the number of bytes in the vacant area of receive buffer is seven or less, sends the Xoff code.
- When the number of bytes in the vacant area of receive buffer is 32 or more, sends the Xon code.

Xon Code	Xoff Code
11H	13H

INPUT

(D), (E) registers Set the channel of RS-232C or RS-422.

OUTPUT

The execution result of SHX is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SHX has been completed normally.
1	Channel error
2	Communication error

NOTE

1. Data sending can be controlled by the Xon/Xoff codes on the external computer or personal computer side.
2. After this subroutine is called, Xon and Xoff codes are placed in 11H and 13H, respectively.
3. This subroutine can be cleared by calling SHD.

SHD

FUNCTION

After called, this subroutine controls the data sending of used channel by the DSR (Data Set Ready) terminal.

DSR State	High Level	Low Level
Execution/non-execution of sending	Execution	Non-execution

INPUT

(D), (E) registers . . . Set the used channel.

OUTPUT

The execution result of SHD is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SHD has been completed normally.
1	Channel error
2	Communication error

NOTE

1. During receiving, control by the DSR terminal is not done.
The receive buffer is of 511 bytes. Control the receive buffer on the user side. (When the number of bytes of receive buffer exceeds 511, error results.)
2. The Xon/Xoff codes are also handled as data.
3. When the power is turned on, SHD is regarded as having been selected.
4. In regards to the input console, the setting switches of main unit have priority over the call of this subroutine. Therefore, when this subroutine is called, channel error results.

SAE

FUNCTION

After called, this subroutine sends and receives the send and receive data after converting them between ASCII code and EBCDIC code.

{ Send data Sent after converted from ASCII code to EBCDIC code.
{ Receive data . . Received after converted from EBCDIC code to ASCII code.

INPUT

(D), (E) registersSet the channel of RS-232C or RS-422.

OUTPUT

The execution result of SAE is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SAE has been completed normally.
1	Channel
2	Communication error

SEA

FUNCTION

Sends and receives the send and receive data to and from the specified channel of RS-232C or RS-422 without the code conversion of send and receive data (ASCII code – EBCDIC code).

INPUT

(D), (E) registers . . . Set the channel of RS-232C or RS-422.

OUTPUT

The execution result of SEA is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SEA has been completed normally.
1	Channel error
2	Communication error

NOTE

When the power is turned on, SEA is selected.

SEN

FUNCTION

Searches the first END instruction located at the specified step and following steps.

INPUT

(D), (E) registers Head step number to be searched

OUTPUT

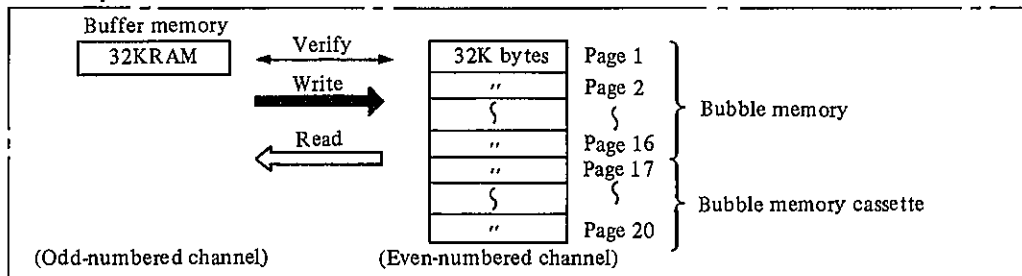
The execution result of SEN is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
END step number	END instruction search has been completed.
FFFFH	No END instruction
FFFEH	Communication error with programmable controller CPU
FFFDH	Specified step error

SBU

FUNCTION

Reads, writes and verifies the memory data between the buffer memory (32K bytes in odd-numbered channel of K30MU10) and bubble memory or bubble memory cassette.



Structure of bubble memory and bubble memory cassette

INPUT

(D) register ← Specified even-numbered channel number (2, 4, 6, 8, A) of bubble memory

(E) register ← Command setting

1	Read	Buffer memory ↔ bubble memory
2	Write	Buffer memory → bubble memory
3	Verify	Buffer memory ↔ bubble memory

(B), (C) registers ← { Specified page of bubble memory (1 ~ 16)
Specified page of bubble memory cassette (17 ~ 20)

OUTPUT

The execution result of SBU is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SBU has been completed normally.
1	Bubble cassette is not loaded.
2	Bubble busy
3	Input setting error
4	Write protect error
5	Communication error between K31MCPU and K30HU10
6	K30MU10 hardware error
7	Verify error

SBE

FUNCTION

Reads bubble error.

When the content of (H) and (L) registers is 1, 4, 6 or 7 after the execution of system subroutine for read, write and verify of bubble memory data (SBU), the content can be checked in more details by SBE.

INPUT

(D), (E) registers ← Even-numbered channel number (2, 4, 6, 8, A) of bubble memory unit to be read.

OUTPUT

(H), (L) registers ← Error data

CAUTION

1. When output is (H)(L) = FFFFH, channel setting error has occurred. Check the channel number and input the number once again.
2. For the error data, see the Instruction Manual of Bubble Memory Unit (K30MU10).

SITX

FUNCTION

Interruption unit data processing

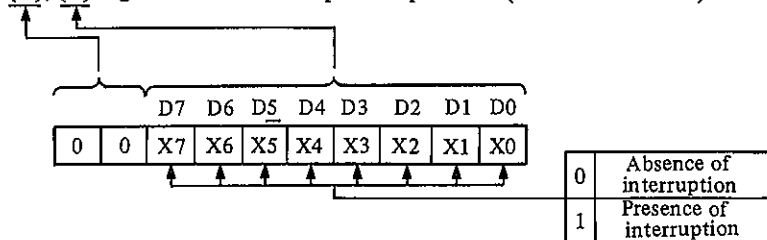
<p>Interruption data is present when this subroutine is called.</p>	<p>Interruption data is absent when this subroutine is called.</p>
<p>Interruption data is stored into (H), (L) registers, and execution returns to user program.</p>	<p>The run of task which has called this subroutine is interrupted and the processing of another task is performed. Then, upon occurrence of interruption input, execution returns to the user program of task which has called this subroutine.</p>

INPUT

(D), (E) registers . . . Even-numbered channel number (2, 4, 6, 8, A) of interruption unit to be specified.

OUTPUT

(H), (L) registers . . . Interruption input data (0001H ~ 00FFH)



CAUTION

1. This subroutine cannot be called by plural tasks for the same channel.
2. After processing the interruption input data read by this subroutine, it is always required to clear the interruption input data by "SITY" in the user program. When the data has been cleared by "SITY", the interruption input that follows becomes effective.
3. When FFFFH is in the (H) and (L) registers, channel error results. (The channel, to which the interruption unit is not loaded, has been specified.)

SITY

FUNCTION

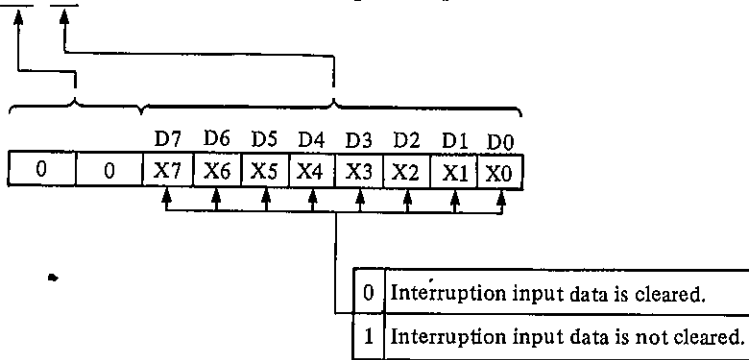
Interruption data clear operation.

Clears the specified interruption input data. (Clears the interruption input data used by SITX.)

INPUT

(D), (E) registers . . . Specified even-numbered channel number (2, 4, 6, 8, A) of interruption unit.

(B), (C) registers Data for clearing in interruption



OUTPUT

The execution result of SITY is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SITY has been completed normally.
1	Specified channel error
4	Clear data error, (B)(C) \geq 100H

EXAMPLE

```
100 A=CALL ($0,$9096,$24,$8000)
```

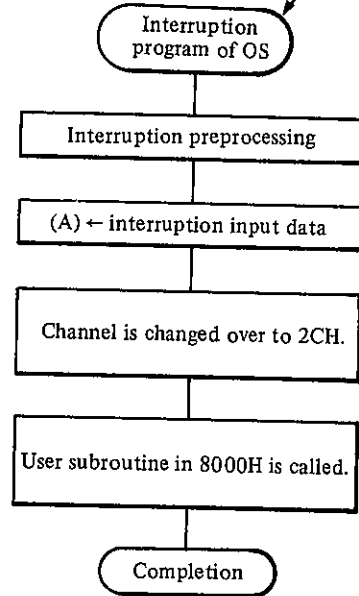
```
110
```

Channel 2

SITA designation

Occurrence of interruption

Interruption has occurred at interruption input unit loaded in channel 4 after above indicated BASIC program has been run.

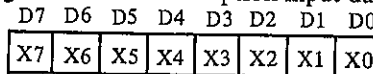


CAUTION

User subroutine for interruption input processing

1. Input condition:

(A) register ← interruption input data



"1": presence of interruption input
"0": absence of interruption input

2. Prepare the user subroutine for interruption input processing in machine language of used CPU.
3. Be sure to end the user subroutine for interruption input processing with "RET".

WARNING

System subroutine commands, which can be used in the user subroutine for interruption input processing, are SMOV, SMOVK, SRD1, SRD2, SWR1 and SWR2. Never use other commands.

For the access to the K3NCPU, do not use SMOV, SMOVK, SRD1, SRD2, SWR1 and SWR2 commands. If these commands are used, the system may become out of control.

SITB

FUNCTION

Clear of interruption input processing subroutine setting

- If interruption occurs at the interruption input unit of specified channel, the user subroutine for interruption input processing set by SITA is not called.
- When interruption occurs after this subroutine is called, interruption input data is set to the interruption input data area, and the call of SITX provides the interruption input data.

INPUT

(D), (E) registers . . . Specified even-numbered channel number (2, 4, 6, 8, A) of interruption input unit.

OUTPUT

The execution result of SITY is stored in the (H) and (L) registers.

(H), (L) Registers	Judgement
0	Execution of SITB has been completed normally.
1	Channel error

NOTE

Channel error occurs when the channel, to which the interruption unit is not loaded, is specified.

4.4 Subroutines Which Can Be Used Only for Microcomputer Program

The following table shows the subroutines which can be used only for the microcomputer program.

System Subroutine	Reference Page	System Subroutine	Reference Page
SCALL	309	STIME	318
SALU	311	SQUIT	319
SMOV	313	SRSFCO	320
SRD1	314	SRSFST	321
SRD2	315	SRSFRD	322
SWR1	316	SRSFWR	323
SWR2	317		

SCALL

TYPE	A
------	---

FUNCTION

Calls a subroutine in the specified address of specified channel.
When calling a subroutine which has been prepared in another channel, it is required to call it indirectly via this system subroutine because the channel should be switched.
Arguments in (D), (E) registers and (B), (C) registers are transferred, as they are, to the call destination subroutine.
Data in (H), (L) registers at the time of RET of call destination subroutine are returned as output, as they are, to the call source program.
Subroutines can be called to a level of five by each task.

INPUT

- (A) register Channel number of subroutine
- (H), (L) registers . . . Head address of subroutine
- (D), (E) registers . . . Argument to subroutine
- (B), (C) registers . . . Argument to subroutine

OUTPUT

- (H), (L) registers . . . Output data from subroutine

WARNING

In OUTPUT, the contents of registers other than the (H) and (L) registers are damaged.
--

SCALL

TYPE	B
------	---

FUNCTION

Calls a subroutine in the specified address of specified channel.
When calling a subroutine which has been prepared in another channel, it is required to call it indirectly via this system subroutine because the channel should be switched.
Arguments in (D), (E) registers and (B), (C) registers are transferred, as they are, to the call destination subroutine.
Subroutines up to a level of five can be called by each task.

INPUT

- (A) register Channel number of subroutine
- (H), (L) registers . . . Head address of subroutine
- (D), (E) registers . . . Argument to subroutine
- (B), (C) registers . . . Argument to subroutine

OUTPUT

Value in each register, except carry flag, at the time of RET of call destination subroutine is returned as output to the call source program.
The execution result of SCALL is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SCALL has been completed normally.
0	Subroutines have been called exceeding a level of five.

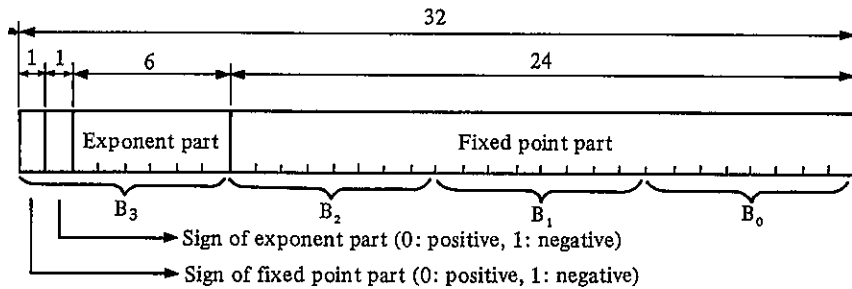
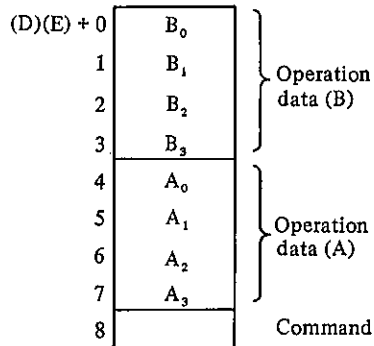
SALU

FUNCTION

Performs the operation of command for which 32-bit floating-point type data have been specified.

INPUT

(D), (E) registers Set the head address of set data.
Set data Set the operation data and command to (D)(E) + 0 to 8.

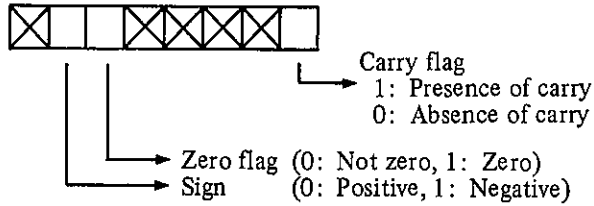


OUTPUT

After the execution of SALU, the contents of carry flag tell which error has occurred.

Carry Flag	Judgement
1	Execution of SALU has been completed normally. (Stores data shown in the following page.)
0	Error

Stored data
(A) register status

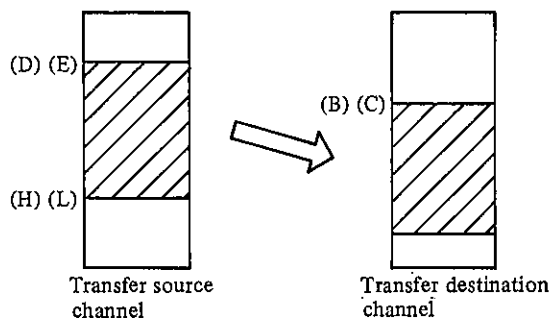


Code (Hexadecimal)	Operation Name	Expression	Description
06	ACOS	$R = \cos^{-1} A$	Arccosine function
05	ASIN	$R = \sin^{-1} A$	Arcsine function
07	ATAN	$R = \tan^{-1} A$	Arctangent function
15	CHSF	$R = -A$	Sign conversion
03	COS	$R = \cos A$	Cosine function
0A	EXP	$R = e^A$	Exponentiation
10	FADD	$R = A + B$	Addition
13	FDIV	$R = B \div A$	Division
1C	FLTD	$A \rightarrow R$	32-bit integer \rightarrow 32-bit floating point
1E	FIXD	$A \rightarrow R$	32-bit floating point \rightarrow 32-bit integer
12	FMUL	$R = AB$	Multiplication
11	FSUB	$R = B - A$	Subtraction
08	LOG	$R = \log_{10} A$	Common logarithm
09	LN	$R = \log_e B$	Natural logarithm
1A	PUPI	$R = \pi$	Circle ratio
0B	PWR	$R = B^A$	Exponent
02	SIN	$R = \sin A$	Sine function
01	SQRT	$R = \sqrt{A}$	Square root
04	TAN	$R = \tan A$	Tangent function

SMOV

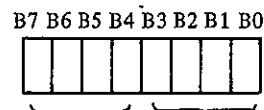
FUNCTION

Transfers data from memory to memory in block.



INPUT

(A) register Set the channels.



(H), (L) registersSet the last address of transfer source.

(D), (E) registersSet the head address of transfer source.

(B), (C) registersSet the head address of transfer destination.

OUTPUT

The execution result of SMOV is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SMOV has been completed normally.
0	Error

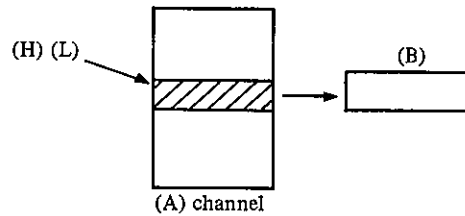
CAUTION

1. Error occurs only when communication with the programmable controller CPU cannot be made.
2. When data is sent and received to and from the programmable controller CPU, STIME is executed for 40 ms in regards to 256 bytes.

SRD1

FUNCTION

Read of one byte from memory.
Reads one byte from the memory of another channel.



INPUT

(A) register Channel from which data is read.
(H), (L) registers Address from which data is read.

OUTPUT

The execution result of SRD1 is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SRD1 has been completed normally.
0	Error

(B) register Read data.

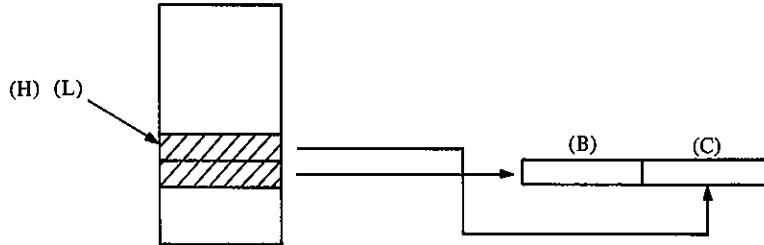
CAUTION

Error occurs only when communication with the programmable controller CPU cannot be made.

SRD2

FUNCTION

Read of two bytes from memory.
Reads two bytes from the memory of another channel.



INPUT

(A) register Channel from which data is read.
(H), (L) registers Address from which data is read.

OUTPUT

The execution result of SRD2 is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SRD2 has been completed normally.
0	Error

(B), (C) registers . . . Read data.

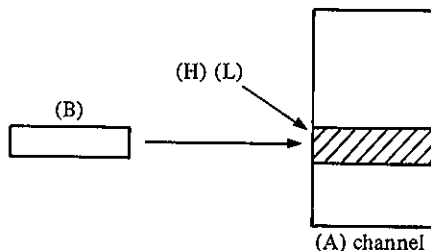
CAUTION

Error occurs only when communication with the programmable controller CPU cannot be made.

SWR1

FUNCTION

Write of one byte to memory.
Writes one byte to the memory of another channel.



INPUT

(A) register Channel to which data is written.
(H), (L) registers . . . Address to which data is written.
(B) register Data to be written.

OUTPUT

The execution result of SWR1 is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SWR1 has been completed normally.
0	Error

CAUTION

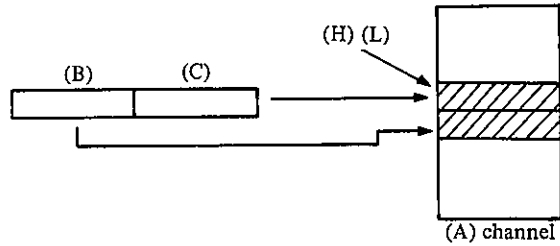
Error occurs only when communication with the programmable controller CPU cannot be made.

SWR2

FUNCTION

Write of two bytes to memory.

Writes two bytes to the memory of another channel.



INPUT

(A) register Channel to which data is written.

(H), (L) registers . . . Address to which data is written.

(B), (C) registers . . . Data to be written.

OUTPUT

The execution result of SWR2 is stored in the carry flag.

Carry Flag	Judgement
1	Execution of SWR2 has been completed normally.
0	Error

CAUTION

Error occurs only when communication with the programmable controller CPU cannot be made.

STIME

FUNCTION

When called, this subroutine interrupts the processing of microcomputer program and runs another task.

After (time limit of timer) x 10 ms has elapsed, execution returns to the STIME call program.

INPUT

(H), (L) registers . . . Time limit of timer (unit: 10 ms)

OUTPUT

None

SQUIT

FUNCTION

End of program run.

Ends the execution of program and returns control to OS.

The same as the END command of BASIC.

INPUT

None

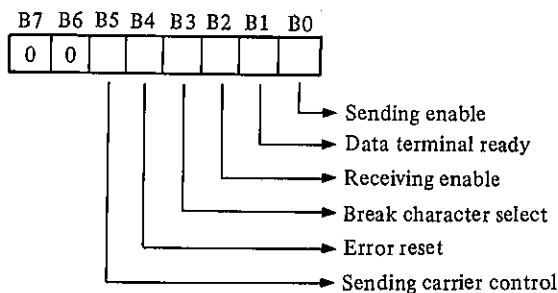
OUTPUT

None

SRSFCO

FUNCTION

Writes a command directly to the programmable communication interface element which is built in the RS-232C interface unit (K30RSF). The structure of the command mode expressed in binary number is as shown below, and each bit is effective when it is 1.



INPUT

- (B) register Channel to which the RS-232C interface unit is loaded.
- (C) register Command

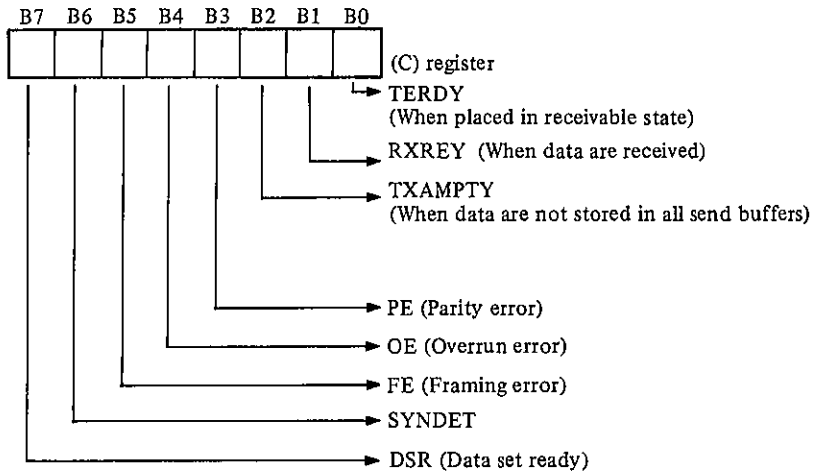
NOTE

For the status, see the Manual of Programmable Communication Interface Element. (8251)

SRSFST

FUNCTION

Reads the status of programmable communication interface element which is built in the RS-232C interface unit K30RSF). The structure of status data is as shown below. Each bit has the following meaning when it is 1.



INPUT

(B) register Channel to which the RS-232C interface unit is loaded.

OUTPUT

(C) register Status of programmable communication interface element.

NOTE
For the status, see the Manual of Programmable Communication Interface Element. (8251)

SRSFRD

FUNCTION

Reads data from the programmable communication interface in the RS-232C interface unit (K30RSF) of specified channel.

INPUT

(B) register Specified channel number

OUTPUT

(C) register Read data.

SRSFWR

FUNCTION

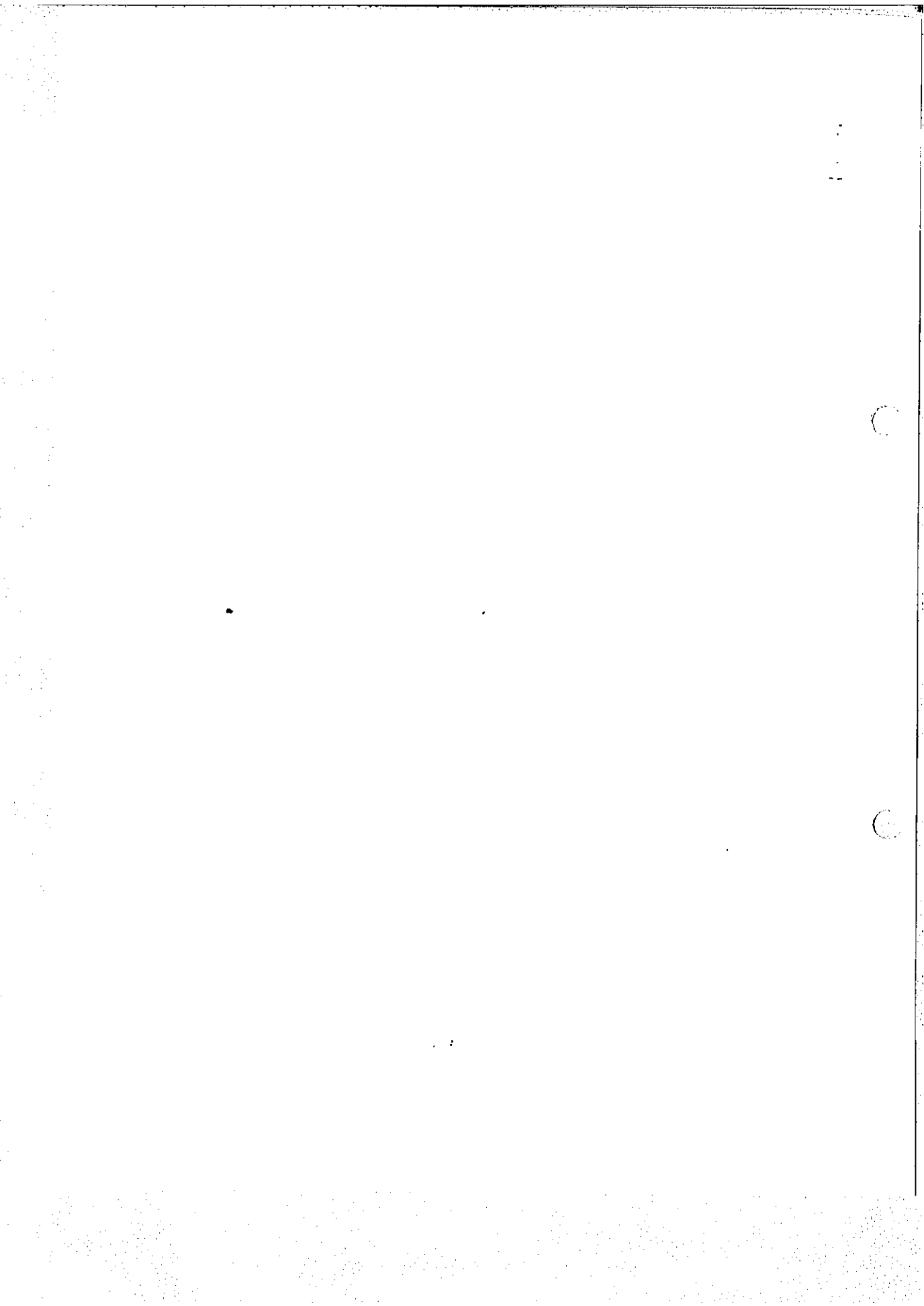
Writes data to the programmable communication interface in the RS-232C interface unit (K30RSF) of specified channel.

INPUT

(B) register Specified channel number
(C) register Data

OUTPUT

None



A decorative rectangular border with a repeating floral or scrollwork pattern surrounds the central text.

APPENDIX

APPENDIX

5.1 Error Messages Displayed during Programming

When program error occurs during BASIC programming, an error message is displayed in the following form:

(Type of error)

(Line number) (Statement)

EXAMPLE

```
OK
>RUN
WHAT?
170 CIRCLE (320, 200), 100.....CIRCLE command in line number 170 is misspelled.
```

Error types are as shown in the following table.

Error Type	Cause
WHAT ?	There is grammatical error. (1) Undefined command has been used. (2) Line number of GOTO, GOSUB, ONGOTO or ONGOSUB command has not been specified. (3) RETURN command has been used although GOSUB or ONGOSUB command is not used.
HOW ?	The same cause as "WHAT ?" Handled numeric value is outside the range of -32767 to 32767.
SORRY	Program capacity has exceeded specified memory area.

CAUTION

1. Error may occur after the calculation result of line number of ONGOTO, ONGOSUB or RETURN command has become a value of non-existing line number. Therefore, caution should be exercised.
2. When the above indicated error occurs during run of multi task, the task number is also displayed, and the execution of system is stopped.

5.2 Cautions during Preparation of BASIC Program

5.2.1 Initial setting during BASIC programming

Table 5.2 shows the contents of the initial screen items which are set during BASIC programming.

Item	Description
1. PROGRAM HEAD ADDRESS	Head address of text of BASIC (Specify address 8000H or address located below 8000H.)
2. PROGRAM LAST ADDRESS	The last address used to secure area as text area of BASIC
3. ADDITIONAL PROGRAM HEAD ADDRESS	Head address when another text of BASIC is inserted.
4. WORK AREA HEAD ADDRESS	Work area used by interpreter of BASIC and fixed to 256 bytes. (This is not a work area used by user.)
5. CHANNEL	Channel where text of BASIC is inserted.

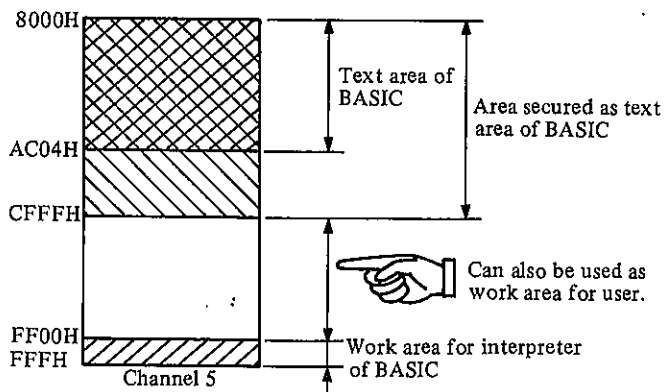
Table 5.2 BASIC Program Addresses

EXAMPLE

The above items have been set as shown below by use of 8000H and the following addresses of channel 5.

1. PROGRAM HEAD ADDRESS 8000
2. PROGRAM LAST ADDRESS CFFF
3. ADDITIONAL PROGRAM HEAD ADDRESS AC05
4. WORK AREA HEAD ADDRESS FF00
5. CHANNEL 5

Set on OS side. When a new program is prepared, this address is the same as the program head address.



CAUTION

1. The basic variable is allotted inside the work area for interpreter of BASIC.
2. The @ array variable and indirect variable can be freely used for the vacant area (addresses D000H to FFFFH) in the above figure. Never use the hatched area.
3. Since the work area for interpreter of BASIC is allotted below the text area and 256 bytes are automatically assigned, be sure to set "OOH" to the lower byte.
4. When preparing two or more tasks in the same channel, the program area and the work area for interpreter should not overlap.

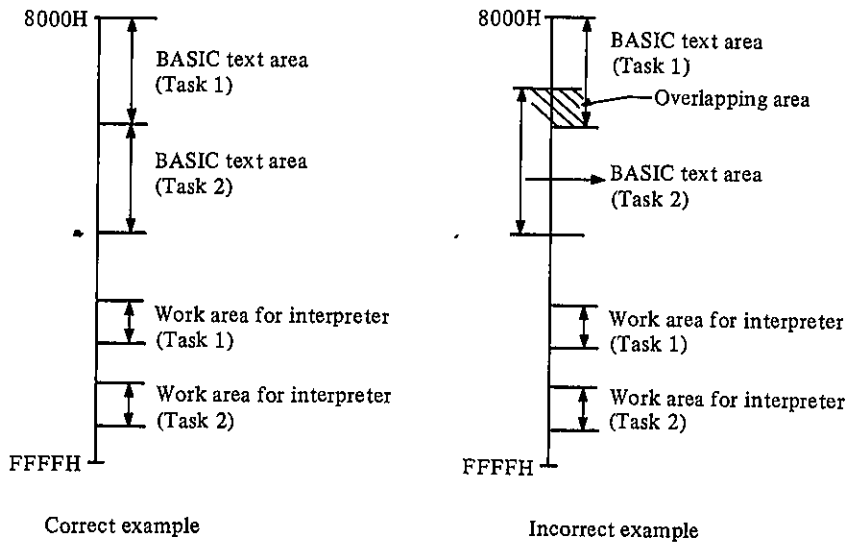


Fig. 5.1 Overlap of Tasks

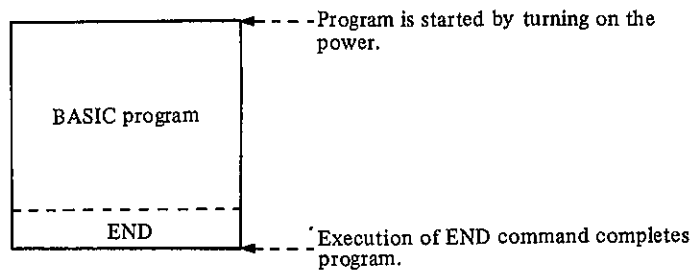
Note that the above caution also applies to the microcomputer program.

5.2.2 Start condition and programming

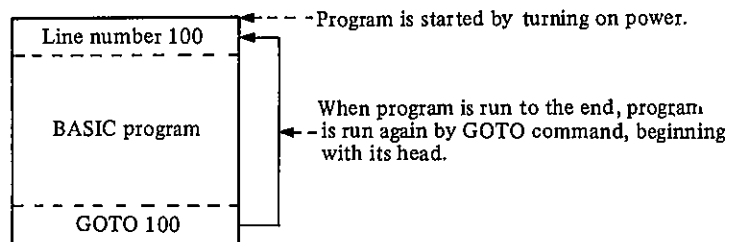
There are the following four types of BASIC program run formats:

- (1) Program is run only once after power-on.
- (2) Program is always run after power-on.
- (3) Program is run by interruption caused by CPU.
- (4) Program is run at set intervals of real time.

- (1) Program is run only once after power-on
Prepare the program so that the END command is executed at the end of program. Select "POWER ON" as the start condition of task.

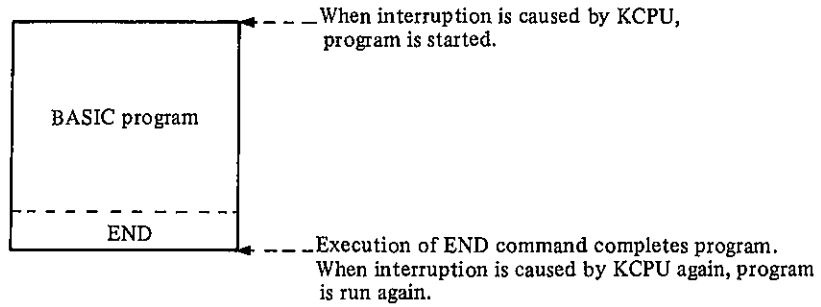


- (2) Program is always run after power-on
Prepare the program so that run is returned to the head of program by the GOTO command, without using the END command at the end of program. Select "POWER ON" as the start condition of task.



(3) Program is run by interruption caused by CPU

Prepare the program so that the END command is executed at the end of program. Select "KCPU INT" as the start condition of task.

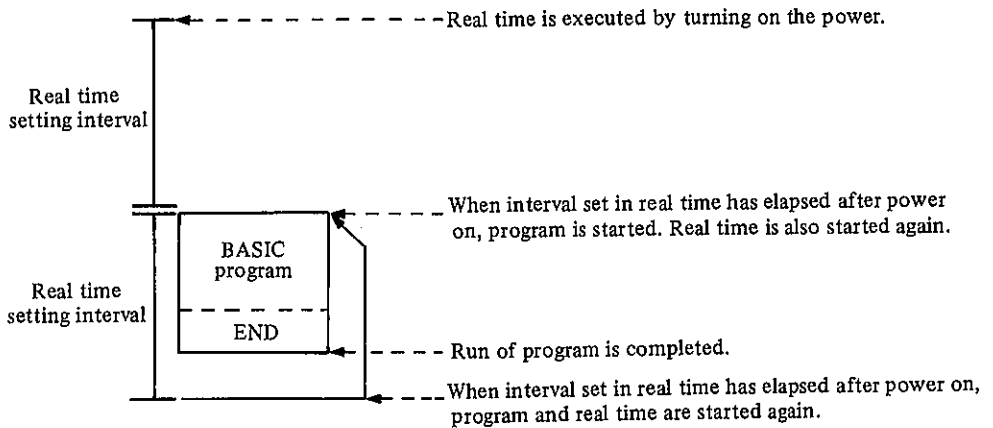


CAUTION

1. The task, which runs the program by interruption caused by the KCPU, should be one. If two or more tasks are provided, "plural ORST occurrence" error may occur. (For details, see Section 5.2.3.)
2. By executing the END command, the task which runs the program by interruption caused by the KCPU resets the interruption caused by KCPU.

(4) Program is run at set intervals of real time

Prepare the program so that the END command is executed at the end of program. Select "REAL TIME INT" as the start condition of task and set the interval.



CAUTION

It is required to set the interval in real time greater than the value obtained by adding the run time of all programs which are started by multi task. If the task, for which "REAL TIME INT" has been selected as the start condition, does not execute the END command within the set interval, "plural ORST occurrence" error will result.

5.2.3 "Plural ORST occurrence"

When there are two or more tasks which are started by interruption caused by the KCPU, the completion of run of one task resets the interruption by KCPU, and the interruption by KCPU occurs again before the run of another task is completed, resulting in "plural ORST occurrence" error.

For this reason, the number of tasks started by the interruption caused by KCPU should be one.

Fig. 5.2 shows an example of error which occurs in the case of tasks 1 and 2 which are started by the interruption caused by KCPU.

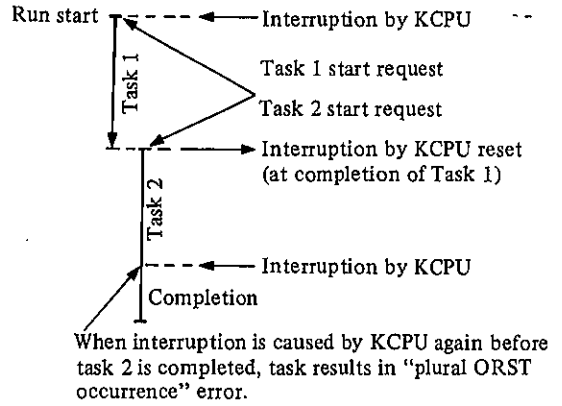


Fig. 5.2 "Plural ORST Occurrence" Error

5.3 Program Prepared with Microcomputer Instruction

The Z80CPU is used as a CPU.

For details of instruction words for preparing the microcomputer program, see the Manual of Z80CPU.

This section describes mainly cautions for preparing a program with microcomputer instruction.

5.3.1 Caution for preparing task

When the program prepared with microcomputer instructions is run as a task, there are the following run formats:

- (1) Program is run only once after power-on.
- (2) Program is always run after power-on.
- (3) Program is run by interruption caused by CPU.
- (4) Program is run by real time interruption.

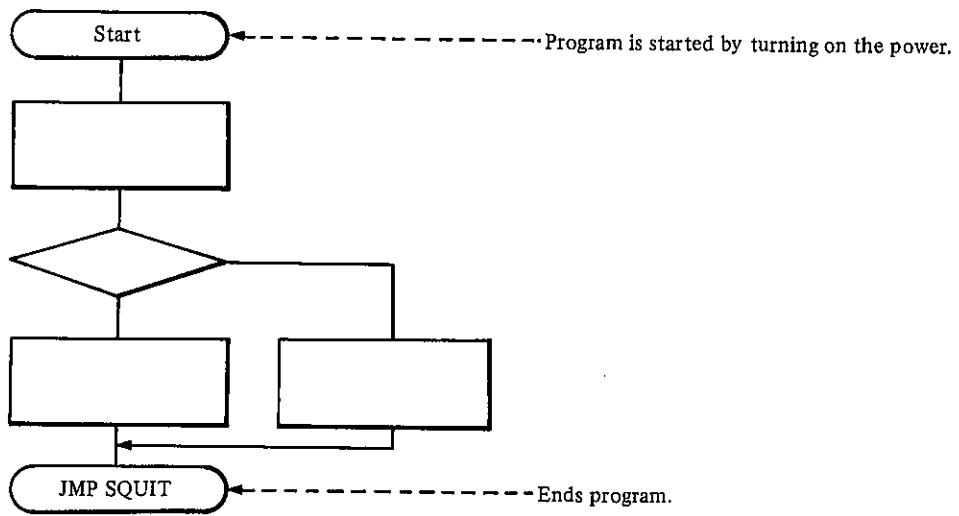
A programming method is explained hereunder.

For the start condition and setting of task, see Instruction Manual of each Model.

- (1) Program is run only once after power-on
Execute JMP SQUIT (JMP 85H) at the end of program. Select "POWER ON" as the start condition of task.

NOTE

JMP SQUIT (JMP 85H) of microcomputer program is equivalent to the END command of BASIC program.

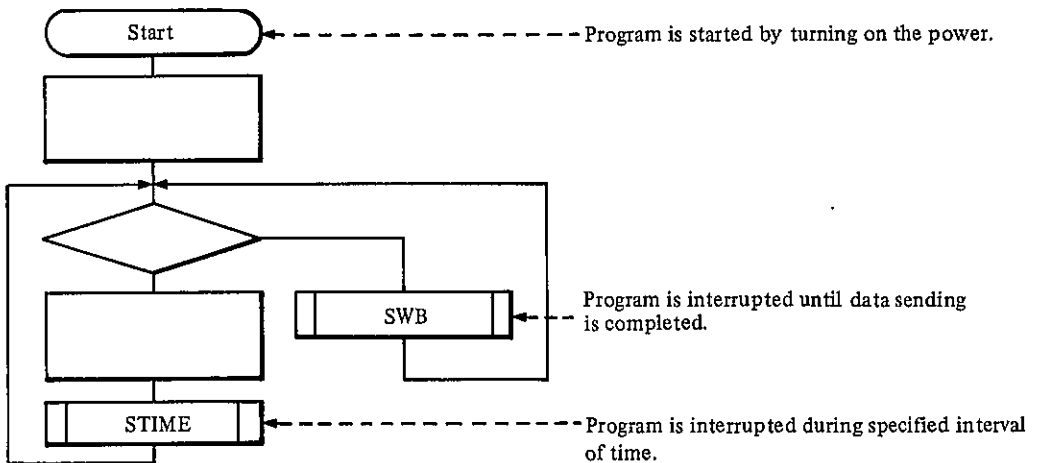


- (2) Program is always run by power-on
 Select "POWER ON" as the start condition of task.
 Prepare the program so that it is looped by use of one of the following system subroutines which change over the task.

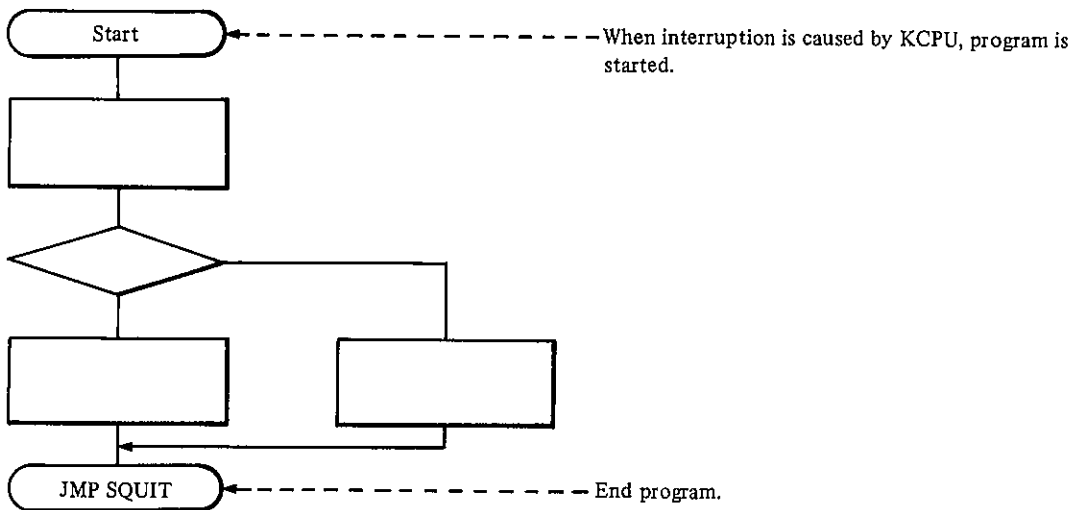
System Subroutine Abbreviation	System Subroutine Abbreviation
SRB	SITA
SWB	SITB
SBU	SMOV
SITX	STIME
SITY	SQUIT

CAUTION

Note that when the microcomputer program is run once, execution is not transferred to another task unless the system subroutine, which changes over the task, is run.



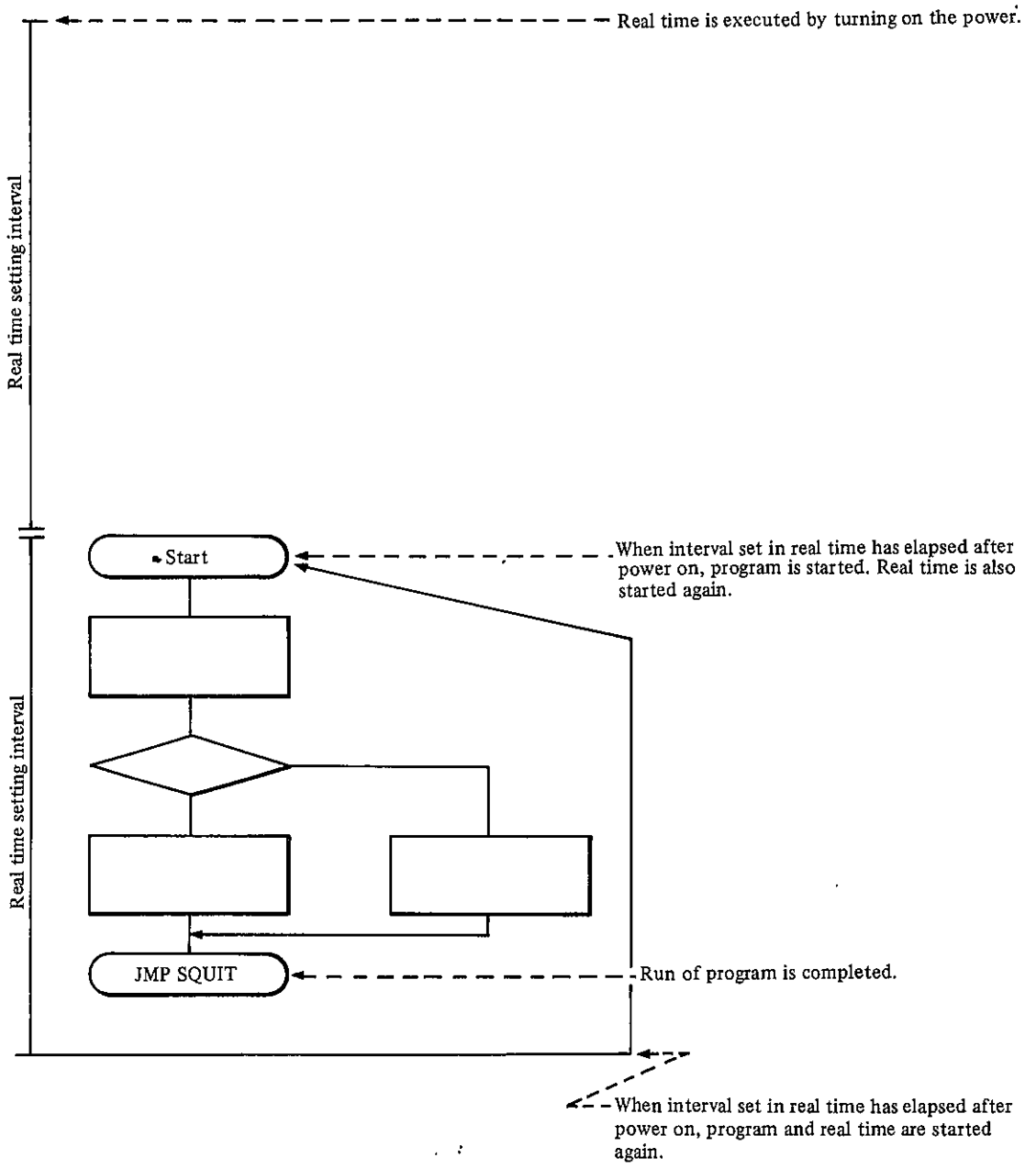
- (3) Program is run by interruption caused by KCPU
 Execute JMP SQUIT (JMP 85H) at the end of program. Select "KCPU INT" as the start condition of task.



- (4) Program is run by real time interruption
 Execute JMP SQUIT (JMP 85H) at the end of program. Select "REAL TIME INT" as the start condition of task.

CAUTION

When "REAL TIME INT" has been selected, prepare the program so that run of all tasks is completed within the specified period of time. If run is not completed within the period, "plural ORST occurrence" error will result. When there is interruption by KCPU or wait for data entry, it is also required to consider its time.



5.3.2 Preparation of subroutine

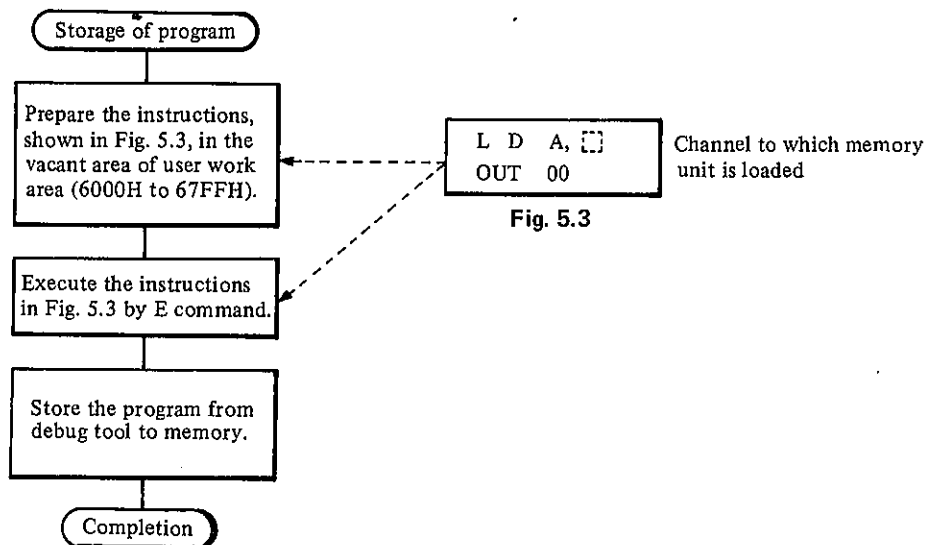
Take care of the following points when preparing a subroutine with microcomputer instruction.

- (1) Subroutine used for BASIC program and microcomputer program in another channel
To call the subroutine, which has been prepared with microcomputer instruction, from the BASIC program, use the CALL command.
To call the subroutine from the microcomputer program, use the SCALL system subroutine.
Note that for this reason, the transfer method of argument is restricted.
- (2) Subroutine used for microcomputer programs in the same channel
To call the subroutine from the microcomputer program in the same channel, use the normal CALL instruction.
Also, the transfer method of argument can be used freely.

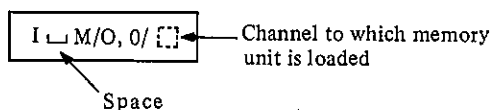
5.3.3 Storing method of microcomputer program

This section shows a procedure for storing the program, which has been prepared with microcomputer instruction, in the memory by use of the debug tool.

- (1) To store a program



- (2) To execute by debug command
Execute the following instruction with the debug tool.



5.3.4 Caution for preparing microcomputer program

- (1) Never use the following instructions. There is a possibility that the system may malfunction.

Instruction Related to Stack	Instruction Related to Interruption	Instruction Related to I/O	Miscellaneous
LD A, I LD A, R LD I, A LD R, A	LD SP, HL LD SP, IX LD SP, IY INC SP DEC SP	IN INI INIR IND INDR OUT OUTI OTIR OUTD OTDR	HALT RST
DI EI			
IM 0 IM 1 IM 2			
RET 1 RETN 1			

- (2) The numbers of PUSHs and POPs should always be the same in the program.
- (3) When it is desired to call a subroutine in another channel, use the SCALL system subroutine.
- (4) Avoid looping for a long time in the program.
When it is desired to loop in the program, suspend program run by use of the system subroutine, which changes over the task, described in Section 5.3 (2). Note that during looping in the program, another task is not run.
- (5) Stack used by the user should be 128 bytes or fewer.

5.4 Addresses of System Subroutines

5.4.1 System subroutines which can be used for BASIC commands

	System Subroutine	KGPC1		KGPC11		KD51		Reference Page
		Channel	Address	Channel	Address	Channel	Address	
1	SAI	—	—	0	9060H	0	8060H	223
2	SIA	—	—	0	9063H	0	8063H	225
3	SAN	—	—	0	9072H	0	8072H	227
4	SNA	—	—	0	9075H	0	8075H	229
5	SAF	—	—	0	9066H	0	8066H	231
6	SFA	—	—	0	9069H	0	8069H	234
7	SBF	—	—	0	906CH	0	806CH	236
8	SFB	—	—	0	906FH	0	806FH	237
9	SBD4	—	—	0	9042H	0	8042H	239
10	SDB4	—	—	0	9045H	0	8045H	240
11	SBD6	—	—	0	9048H	0	8048H	241
12	SDB6	—	—	0	904BH	0	804BH	243
13	SBA	—	—	0	904EH	0	804EH	245
14	SBS	—	—	0	9051H	0	8051H	247
15	SBM	—	—	0	9054H	0	8054H	249
16	SBW	—	—	0	9057H	0	8057H	251
17	SCA	—	—	0	9039H	0	803CH	253
18	SCB	—	—	0	903CH	0	803FH	254
19	SMOVK	—	—	0	902AH	—	—	255
20	BBO	FH	8006H	—	—	—	—	257
21	BCR	FH	8000H	—	—	—	—	259
22	BWK	FH	8003H	—	—	—	—	261
23	SDE	FH	A9H	—	—	—	—	263
24	SLD	FH	A6H	—	—	—	—	264
25	SLE	FH	8033H	—	—	—	—	266
26	SOK	FH	AFH	—	—	—	—	267
27	SST	FH	A3H	—	—	—	—	268
28	SPC	—	—	—	—	0	8078H	270
29	SRK	—	—	—	—	0	8024H	271
30	SWK	—	—	—	—	0	8027H	273
31	SRI	—	—	—	—	0	802AH	275
32	SWI	—	—	—	—	0	802DH	277
33	SKC	—	—	—	—	0	8030H	279

NOTE: H at the end of channel and address indicates the number is hexadecimal.

	System Subroutine	KGPC1		KGPC11		KD51		Reference Page
		Channel	Address	Channel	Address	Channel	Address	
34	SKR	--	--	--	--	0	8033H	280
35	SKP	--	--	--	--	0	8036H	281
36	SKI	--	--	--	--	0	8039H	282
37	SOPEN	FH	801BH	0	901BH	--	--	283
38	SRB	A	FH	8009H	--	--	--	285
		B	--	--	0	9009H	0	8009H
39	SWB	A	FH	800CH	--	--	--	289
		B	--	--	0	900CH	0	800CH
40	SRC	--	--	0	900FH	0	800FH	294
41	SRF	--	--	0	9012H	0	8012H	295
42	SHX	--	--	0	9015H	0	8015H	296
43	SHD	--	--	0	9018H	0	8018H	297
44	SAE	--	--	0	901EH	0	801EH	298
45	SEA	--	--	0	9021H	0	8021H	299
46	SEN	--	--	--	--	0	801BH	300
47	SBU	--	--	0	9030H	--	--	301
48	SBE	--	--	0	9033H	--	--	302
49	SITX	--	--	0	9090H	--	--	303
50	SITY	--	--	0	9093H	--	--	304
51	SITA	--	--	0	9096H	--	--	305
52	SITB	--	--	0	9099H	--	--	306

5.4.2 System subroutines which cannot be used for BASIC commands

	System Subroutine	KGPC1		KGPC11		Reference Page
		Channel	Address	Channel	Address	
1	SCALL	A	—	49H	—	309
		B	—	—	49H	310
2	SALU	—	40H	—	40H	311
3	SMOV	—	76H	—	76H	313
4	SRD1	—	88H	—	88H	314
5	SRD2	—	8BH	—	8BH	315
6	SWR1	—	9DH	—	9DH	316
7	SWR2	—	A0H	—	A0H	317
8	STIME	—	94H	—	94H	318
9	SQUIT	—	85H	—	85H	319
10	SRSFCO	—	—	—	31H	320
11	SRSFST	—	—	—	34H	321
12	SRSFWR	—	—	—	2EH	323
13	SRSFRD	—	—	—	2BH	322

5.4.3 Work area for system subroutine

	Work Area Name	Address	Byte Length	Used Subroutine
1	B C R W	5080H	8	B C R
2	B W K W	50A0H	24	B W K
3	B B O W	50C0H	24	B B O

With this GPC-BASIC handbook, Mitsubishi Electric Corporation does not warrant the enforcement of industrial property and other rights nor grants licenses. Also, Mitsubishi Electric Corporation disclaims all the responsibility for problems on the industrial property attributable to the use of the contents of this GPC-BASIC handbook.

Specifications subjects to change without notice. .

GPC-BASIC

 **MITSUBISHI ELECTRIC CORPORATION**

HEAD OFFICE: MITSUBISHI DENKI BLDG. MARUNOUCHI, TOKYO 100. TELEX: J24532. CABLE: N'ELCO, TOKYO.