1. GENERAL

This reference manual describes the functions and commands of HARL-III (Hirata Assembly Robot Language-III).

HARL-III is a superior robot language for Hirata's own AR & CR series assembly / handling robots as well as the whole manufacturing systems, and has the functions to control entire work cell efficiently.

FEATURES OF HARL-III

(1) Superset of Quick BASIC with enhancements for Factory Automation or CIM

(2) Superior data processing capability

1) Operation (calculation, real number)

2) Function (arithmetic functions, literal function)

3) Several data forms (integer number, real number, string, array)

4) Communication (User can set protocol and communication speed)

(3) Interruption function

Error interruption

(4) High level robot control

1) Enhanced robot motion control

2) Sequence command

2. BASIC FUNCTIONS OF HARL-III

2.1 Job

In HARL-III, operations and functions of a robot and its peripheral equipment are divided into units, and a program is written for each job. Various synchronous and synchronous control programs for these jobs are processed concurrently in the multi job method. (Max 32 jobs)

Robot control job	Pallet positioning control job	Parts supply: control job	Operation: control job

(Max 32 jobs)

Fig. 2.1

By the multi job method, a program can be structured by dividing the jobs according to the functions and/or devices. Therefore, a program would be more simplified and can be packaged to form a job. You can create libraries of application systems easier.

(Library)

Fig. 2.2

2.1.1 Features of Job

Amount of jobs, steps, labels, definitions, local and global variables;

Label	: 1500 Labels / job
Definition	: 1500 definitions
local variable	: 200 variables / job
global variable	: 300 variables of all jobs
network variable	: 100 variables of all jobs

32 jobs (max), about 10,000 to 13,000steps (total amount) are possible to program.
No dedicated job.

There were special jobs and commands in HARL-III, Auto initialise program, manual initialise program and manual program. However, there are no dedicated jobs and commands in HARL-III.

Therefore, error processing or job enabling/disabling does not have to be centralised into one job program and you can design your own job structure.

2.1.2 Starting of all Jobs

When the controller power is turned on, all the job programs of HARL-III are activated at the same time. If a job must be started only upon receiving START command from another job, halt the job execution at the top of its program.

The program text for a job starts from JOB NAME command and ends at the last line or before the next JOB NAME command.

Basically, each job is separated individually and does not interfere with each other. However, the following are the common functions among all jobs and these functions can be used as the interface between each jobs.

Memory ....(See section 2.2)

Timer .... (See section 2.3)

TIME$ and DATE$ .... (See section 3.7 (4))

Global variable number.. (See section 3.8)

JOB START / ON / OFF .... (See section 4.2.2)

File number, communication port number ....(See section 2.4)

2.2 Memory

The variables reserved in the system are called memories.

In HARL-III, the following memories can be used as variables.

<Memory in STP-III>

Memory type	**Format**	**Memory address**	**Description**
Bit memory	MBm	m = 0 to~55	1 bit status memory
Data memory	MDM	m = 0 to 255	1 byte status memory
Word memory	MWm	m= 0 to 255	2 byte status memory
Bit input port	INBm	m= 0 to 255	1 bit input port memory
Data input port	INDm	m= 0 to 31	1 byte input port memory
Bit output port	OUTBM	m = 0 to 255	1 bit output port memory
Data output port	OUTDm	m = 0 to 31	1 byte output port memory
Position memory	Pm	m= 0 to 607 The applicable range is defined by user	Position data memory
Position X component	PXm		X component of Pm
Position Y component	Pym		Y component of Pm
Position Z component	PZm		Z component of Pm
Position W component	PWm		W component of Pm
Position ARM component	ARMm		ARM component of Pm
Position M component	PDMm		M-code Data component of Pm
Position F component	PDFm		F-code data component of Pm
Position S component	Sm		S-code data component of Pm

Table 2.1

Note:

1. The "m" range of Pm must be defined by DIMPOS command.

2. In case of indirect designation with other memories or numeric expression, enclose the memory number in parentheses.

3. Even if the power of the controller is reset, the contents of MD, MB, MW, P and the components of P are not initialised.

Example)

(1) MB(MD 5): 1 bit status memory whose number is specified by the content of MD 5.

(2) P(A+10): Position memory whose number is specified by the variable "A" plus 10.

<memory in robot controller (HNC) >

Memory type	**Format**	**Memory address**	**Description**
Robot position memory	PMm	m = 0 to 999	Robot position data memory
M data	MMm		M data of PMm
Speed data	FMm		Speed data of PMm
Auxiliary data	SMm		For special functions
Robot bit memory	MRBm	m = 0 to 63	1 bit status memory of robot
Robot data memory	MRDm	m = 0 to 7	1 byte status of robot memory
Robot bit output port	ORBm	m = 0 to 31	1 bit output port memory of robot
Robot data output port	ORDm	m = 0 to 31	1 byte output port memory of robot
Robot bit input port	IRBm	m = 0 to 31	1 bit input port memory of robot
Robot data input port	IRDm	m = 0 to 31	1 byte input port memory of robot
Robot status	STATUSm	m = 0 to 9	Status information of robot
Robot current position	HERE		Current position data of robot

Table 2.2

Note:

1. When using these memories, REF command must be used.

2. In case of indirect designation with other memories or numeric expression, enclose the memory number in parentheses.

Example)

(1) IRB(MD5): 1 bit input port memory whose number is specified by the content of MD5.

(2) PM(A+10): Robot position memory whose number is specified by the variable "A" plus 10.

<Memory Pointer>

You may want to specify a memory using the contents of another memory, variable, or formula instead of a straight integer value as mentioned above. Such a variable, memory, or formula is called as Memory Pointer. You can put it right after a memory name within parentheses.

2.2.1 MB, MD

MB: There are 256 bit memories (MB) 0 through 255, and each can store the value of 0 or 1.

MD: There are 256 data memories (MD) 0 through 255, and each can store a positive integer value 0 through 255. Data memories 0 through 31 correspond to bit memories (MB) 0 through 255, so that bit information can be handled in byte unit.

For example, MD0 consists of 8 bit MB0 through MB7. The relation is shown below.

MD0	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

These MBs are assigned the value by 8 bit binary as the contents (0 through 255) of MD.
When the value of MD0 is 0, 150 or 255, the value of MB is shown below.

	MSB							MLB
(The Value of MD0)	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
0	0	0	0	0	0	0	0	0
150	1	0	0	1	0	1	1	0
255	1	1	1	1	1	1	1	1
	128	64	32	16	8	4	2	1

Each weight of MB corresponding to the value of MD

Example) When the value of MD0 is 150.

150=128x1+64x0+32x0+16x1+8x0+4x1+2x1+1x0

The explanation above shows the relation between MD0 and MB0 to MB7. The same relation is applied between MB0 to 255 and MD0 to 31 as shown in table 2.3.

The MD253 to 255 (data memory) are assigned to monitor the state of field-bus network. Do not use these MD as your memory.
See "2.2.9 How to Refer to Field-bus State from HARL-III Program" about MD253 to 255.

Note:

1. The number of each memory can be specified indirectly by numeric expression or the value of another memory called as "pointer". The memory pointer must be enclosed in the parentheses.

2. Even if the power of the controller is reset, the contents of MB and MD are not initialised.

<Correspondence between MD and MB>

MD	MB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	134	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
32	128	64	32	16	8	4	2	1
..... 255	Each weight of MB corresponding to the value of MD							

Table 2.3

2.2.2 MW

MW: There are 256 word memories (MW), and each can store an integer value - 32768 through + 32767. MWs occupy an independent memory area. There is no overlapping like between MDs and MBs.

Even if the power of the controller is reset, the contents of MW are not initialised.

The MW254 to 255 (data memory) are assigned to monitor the state of field-bus control. Do not use these MW as your memory.
See "2.2.9 How to Refer to Field-bus State from HARL-III Program" about MW254 to 255.

2.2.3 INB, IND, OUTB, OUTD

INB: There are 256 input bit memories (INB) INB0 through INB255 and each can store the value 0 or 1 in order to express on/off status of input signal from external devices.

IND: There are 32 input data memories (IND) IND0 through IND31 in order to process consecutive 8 INBs by byte units.

Note:
Some signals from external devices may be intervened by electric noise or chattering. (Pulse signal which is occurred by bound between contacts.) After removing the chattering time by software, input signals are processed as INB or IND. Therefore, there is 20 msec delay (max) until a signal is stored into a memory.

Same as M13, IND0 consists of 8 bits from INB0 through INB7 and the relation is shown below.

IND0	INB 7	INB 6	INB 5	INB 4	INB 3	INB 2	INB1	INB0

The same relation is applied between INB0 to 255 and IND0 to 31 as shown in Table 2.5.

OUTB: There are 256 output bit memories (OUTB) OUTB0 through 255 and each can store the value 0 or 1 in order to turn on (1) or off (0) the output signal to an external device.

OUTD: There are 32 output data memories (OUTD) OUTD0 through 31 and each can process consecutive 8 OUTBs as I Byte output data.

Same as MD, OUTD0 consists of 8 bits from OUTB0 through 7 and then relation is shown below.

OUTD0	OUTB7	OUTB6	OUTB5	0UTB4	OUTB3	OUTB2	OUTB1	OUTB0

The same relation is applied between OUTB0 to 255 and OUTD0 to 31 as shown in Table 2.4.

Table 2.4 shows the relation between INDs and INBs, OUTDs and OUTBs.

IND/ OUTD	**INB/OUTB**							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	134	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
	128	64	32	16	8	4	2	1
	Each weight of INB /OUTB corresponding to the value of IND / OUTD							

Table 2.4

Note:

Practically the number of input/output depends on the hardware structure such as the number of remote I/O units connected to a HAC robot controller. The address of each memory can be specified indirectly by numeric expression or another memory in parentheses.

2.2.4 P and Component of P

P: P is the memory to handle the position data of robot. P can not be used until the total amount of P is defined. The maximum numbers available by P is 608 (P0 through P607).

Example)

When 20 position memories are required, define the position memories by DIMPOS command.

DIMPOS 20

Then, 20 memories from.P0 through P19 can be used. If the amount of position memories defined by DIMPOS command is different among the jobs, the memories defined in each jobare applicable in the job.

A P consists of 8 components as below.

	Pm						
PXm	PYm	PZm	PWm	ARMm	PDMm	PDFm	PDSm
X component	Y component	Z component	W component	ARM component	M data component	F data component	S data component

"m" represents P number. In case of indirect designation using a variable or expression as a pointer instead of an integer, enclose such a pointer with parentheses.

PX, PY, PZ, PW:	Each of them is a memory for the component of X, Y, Z and W and these are long word (4 byte) memories which can store the value - 2147483.648 through + 2147483.647.
ARM:	ARM is the memory for the component of P and can store LEFTY or RIGHTY. ARM component is available for SCARA type robot mechanism.
PDM:	PDM is the memory for the M-code data which can store value 00h to 99h and ??= (&HFF)
PDF:	PDF is the memory for the F-code data which can store value 00 to 99
S:	PDS is the memory for S-code data which can store value 00 to 99. There is no direct access to the S data of P

Even if the power of the controller is reset, the contents of P and its components are not initialised. It has battery backup.

[How to use P]

Each data of P components is created and used with REF command like the samples below.

Example 1)

Substitute a robot position memory (PM) for a P.

```
P10 = REF (#1, PM110)
```

After the program above has been executed, the following data are substituted for each component of P10.

```
PX10  .........X Axis data of PM110
PY10  .........Y Axis data of PM110
PZ10 ..........Z Axis data of PM110
PW10  ........W Axis data of PM110
ARM10  ......L or R data of PM110
PDM10  ......M data of PM110
PDF10  .......F data of PM110
```

S code of PM110 are substituted for the internal area of P10.

PX10, PY10, PZ10 and PW10 can be handled as numerical variables in the program.
ARM10 can be specified by ARM10 = RIGHTY or ARM10 = LEFTY.
PDM10 can be specified by a value between 00 and 99, or 255 = end point.
PDF10 can be specified by a value between 00 and99.

The M data, F code and S code of PM110 are substituted for the internal area of P10, but those data in P10 can not be write directly. When setting those data, refer to Example 4) below.

Example 2)

Substitute the current position data of the robot for a P.

```
P0 = REF (#1, HERE)      Substitute the current position for P0.
PZ0 = PZ0 - 10           Set Z Axis data of P0 at 10 mm upper than the
                         current position.
PW0 = 200                Set 200 degree as W Axis data of P0.
MOVE #1, PTP, P0         Move the Z, W Axes to the modified position.
```

* When the current position of the robot is substituted, M=01, F=99 and S=00 are automatically stored in the internal area of the P.

Example 3)

Substitute the robot current position data of the robot for a PM.

```
P0 = REF (#1, HERE)         Substitute the current position for P0.
REF (#1, PM100) = P0         Substitute the data in P0 for PM100.
```

* When the program above is executed, MM=01, FM=99 and SM=00 are stored in PM100.

When setting a specific data to MM of PM100, program like the sample below.

```
REF (#1, MM100) = 50 Set 50 as M data of PM100.
```

Example 4)

Copy the robot position memory PM to another PM.

```
 P0 = REF (#1, PM100)      Substitute the data of PM100 for P0.
 P1 = REF (#1, PM200)      Substitute the data of PM200 for P1.
 PX1 = PX0                 Copy X, Y, Z, W and ARM data of P0 to P1.
 PY1 = PY0
 PZ1 = PZ0
 PW1 = PW0
 ARM1 = ARM0
 PDM1=PDM0
 PDF1=PDF0
 REF (#1, PM200) = P1      Substitute P1 data for PM200.
```

By the program above, X, Y, Z, W and ARM data of PM100 are copied to PM200, but the S code stored in PM200 remains at PM200.

2.2.5 PM, MM, FM, SM

PM: PM is a position memory stored in the robot controller. One PM consists of the components below.

PMm							
X component	Y component	Z component	W component	ARM component	MM component	FM component	SM component

Each component of PM can not be handled directly. When setting each component in the program, use REF command to substitute the data for P. Refer to section 2.2.4.

MM: MM is a M data stored in the robot controller. This memory can store the value from 0 through 99. However, the value of MM at end point (??) is assigned as 255.

The MM data can be substituted by REF command.

MD1 = REF (#1, MM1)	MM data of PM1 is substituted for MD1.
REF (#1, MM1) = MD1	The value in MD1 is substituted for MM of PM1.

FM: FM is a F code stored in the controller. This memory can store the value from 0 through 99. The value of FM can be retrieved, but not substituted.

MD1 = REF (#1, FM1)	Applicable to retrieve
REF (#1, FM1) = MD1	Not applicable to substitute

SM: SM is a S code stored in the controller. This memory can store the value from 0 through 99.

Note:

The value of "m" can be specified indirectly with a Memory Pointer.

2.2.6 MRB, MRD, ORB, ORD, IRB, IRD

MRB: There are 64 bit memories from MRB0 through 63, stored in a robot controller. Each can store the value 0 or 1.

MRD: There are 8 data memories, from MRD 0 through 7, stored in a robot controller, and each memory can store the value from 0 through 255. Each of them is a 1 byte (8 bit) memory. Consecutive 8 MRBs are handled as one MRD.

ORB: There are 32 output memories from ORB0 through 31 and each can store the value 0 or 1 in order to turn on (1) or off (0) the parallel output signal of each robot controller. The actual available output bits depend on the hardware structure.

ORD: There are 4 output data memories from ORD0 through 3. Each can process consecutive 8 ORBs as 1 byte parallel output data.

IRB: There are 32 bit input memories from IRB0 through 31. Each can store the value 0 (off) or 1 (on) in order to express on/off condition. The actual available input bits depend on the hardware structure.

IRD: There are 4 input data memories from IRD0 through 3. Each can handle consecutive 8 IRBs as 1 byte parallel input data.

The relation of MRD and MRB, ORD and ORB, IRD and IRB are shown below.

MRD ORD IRD	MRB ORB IRB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4*	39	38	37	36	35	34	33	32
5*	47	46	45	44	43	42	41	40
6*	55	54	53	52	51	50	49	48
7*	63	62	61	60	59	58	57	56

The * marked are only available for MRD. Each weight of MRB, ORB, IRB corresponding to MRD, ORD, IRD.

Table 2.5

2.2.7 Status

There are 10 memories (read only) from STATUS 0 through 9 (STATUS7 is not used) in order to show the conditions of a robot such as status data or error data. The status data always keeps the latest information. Each STATUS can be read by REF command.

<STATUS 0>

STATUS 0 shows one of the error codes (hexadecimal) below. Refer to section 5.5

Error code	**Description**
&H00	normal condition
&H10	Emergency stop condition
&H11	Deadman switch is on.
&H20	A-CAL incomplete (See STATUS 1 to 6)
&H21	Positioning does not complete.
&H30	Address is out of limit.
&H31	M data is not normal.
&H32	W Axis Sensor Stop does not work.
&H33	Data for free curve movement is not normal.
&H34	EPI retry error
&H40	Position data is out of limit area (See STATUS 1 to 6)
&H51	Overrun of robot (See STATUS 1 to 6)
&H60	Communication format error
&H61	Communication command error
&H62	Received a command which is not available in the mode.
&H63	System data (SG, SP) are destroyed.
&H64	Position data are destroyed.
&H65	ON-LINE communication error
&H66	Watch dog time out error *
&H72	Servo error (See STATUS 1 to 6.)
&H80	Received a command while executing another command.
&H90	Moving distance is too short.
&H91	Pass motion PTP overflow error
&H92	Pass motion PTP underflow error
&H93	Over speed error
&H94	M number is not proper
&H95	Coordinates conversion error
&H96	Final positioning can not be completed.
&H99	Motor does not work.
&HA0	Abnormal status at servo driver
&HB0	Abnormal status at encoder line

Table 2.6

<STATUS 1 to 6>

STATUS 1 to 6 store the error data of each axis. Those memories are applicable when the error code of STATUS0 is related to the robot motion. (Error code &H20, &H40, &H51 or &H72 in Table 2.6.)

STATUS 1	Error data of X axis
STATUS 2	Error data of Y axis
STATUS 3	Error data of Z axis
STATUS 4	Error data of W axis
STATUS 5	Error data of 5th axis
STATUS 6	Error data of 6th axis

Table 2.7

(1) When the error code of STATUS0 is &H20, A-CAL error is stored by the value 0 to 7 as error data of each axis to STATUS 1 to 6. (Refer to operation manual of the robot.)

(2) When the error code of STATUS0 is &H40, the area error is stored by the value 0 to 2 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Lower side, 2: Upper side)

(3) When the error code of STATUS0 is &H51, overrun is stored by the value 0 to 3 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Origin, 2: Overrun side, 3: Both)

(4) When the error code of STATUS0 is &H72, servo error is stored by the value 0 to 1 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Error)

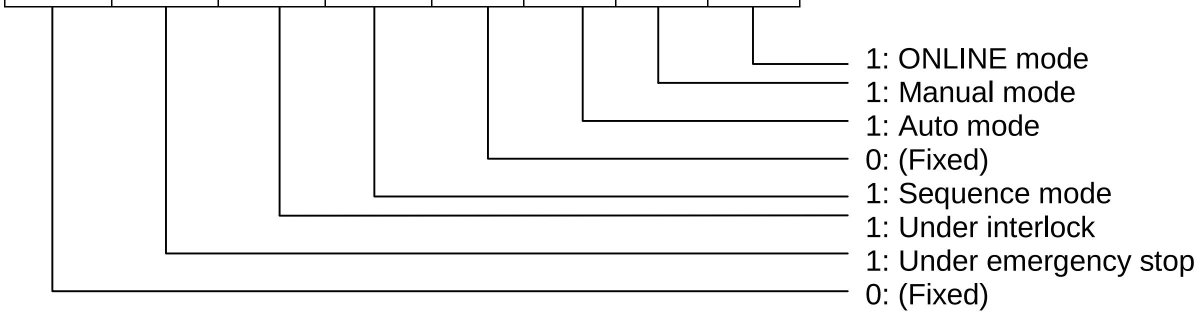
For example, if the error code of STATUS0 is &H51, one or some of the robot axes is in overrun status. When the data of STATUS2 is 1 at that time, Y Axis is in overrun at its origin side.

STATUS 8, 9 consist of 8 bit flags, and these flags are assigned to check the mode or status of the robot.

Each flag has the "weight ," and the total amount of flag weight which are turned on (1) is the value of STATUS m.

<STATUS 8>

f7	f6	f5	f4	f3	f2	f1	f0
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1



<STATUS 9>

f7	f6	f5	f4	f3	f2	f1	f0
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1

f0	1: Under Z safety zone
f1	1: Positioning completion
f2	1: A-CAL completion
f3	0: (Fixed)
f4	0: (Fixed)
f5	0: (Fixed)
f6	1: Under command execution
f7	0: (Fixed)

[How to use STATUS8, 9]

STATUS data can be retrieved by REF command like the example below.

```
* LOOP

IF (REF (#1, STATUS8) AND &H1) <> &H1 THEN GOTO *LOOP
```
(If the mode on the Teach Pendant is not in ON-LINE, the program execution is looped.)

```
MD0 = REF (#1, STATUS9)
```
(Substitutes the value of STATUS9 for MD0.)

```
IF (MD0 AND &H4) = &H4 THEN GOTO *CALIB. OK
```
(If A-CAL has been completed, jump the program execution to CALIB. OK.)

Before using REF command to read STATUS, the communication port should be opened by OPEN "COM command in advance.

2.2.8 HERE

HERE: HERE is a position memory which always keeps the current and actual position data of the robot. The components are shown below.

HERE				
X component	Y component	Z component	W component	ARM component

2.2.9 How to Refer to Field-bus State from HARL-III Program

Outline

The status of the field-bus and a error can be referred to by using MD253, MD254, MD255 and MW254, MW255 from the HARL-III program. Stored contents vary in the type of the field-bus module that it is attached to the STC board.
These memories are always written by STC OS, and it is hatched. Therefore, it can't be used as a user's variable area.
These functions are effective after the STC ROM version 5.40.
The contents of each memory are shown in the following.

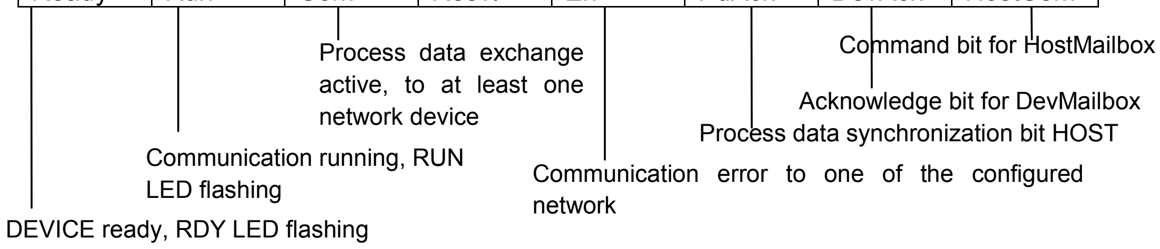
MD253

COM-IBSM

D7	D6	D5	D4	D3	D2	D1	D0
Ready	Run	Com	Resv.	Resv.	PdAck	DevAck	HostCom

Except for COM-IBSM

D7	D6	D5	D4	D3	D2	D1	D0
Ready	Run	Com	Resv.	Err	PdAck	DevAck	HostCom


Command bit for HostMailbox
Acknowledge bit for DevMailbox
Process data synchronization bit HOST
Process data exchange active, to at least one network device
Communication running, RUN LED flashing
Communication error to one of the configured network
DEVICE ready, RDY LED flashing

- HostFlags (1byte) that it is informed from a field-bus module is stored.
- When an error occurs in disconnection of line and so on, a Run bit turns it off in case of COM-IBSM. Therefore, an error can be detected by the following program.

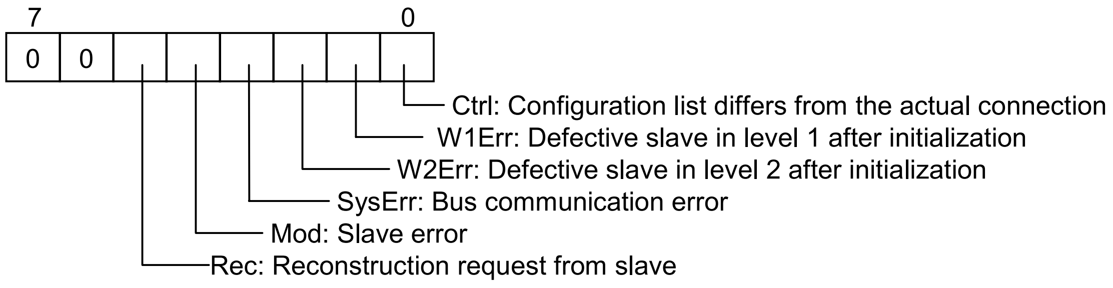
```
          IF (MD253 AND &H40) = 0 THEN GOTO *IOERROR
```

- Except for COM-IBSM, both Run and Ready always become ON except during reset of the field-bus module.
  But, when Run is OFF in the case of the bad configuration set up in the field-bus module.
  Com becomes OFF when the field-bus module cannot exchange I/O data because of the disconnection of line and so on.
  An error can be detected by the following program from the above thing.

```
          IF (MD253 AND &HE0) = 0 THEN GOTO *IOERROR
          IF (MD253 AND &H08) = 1 THEN GOTO *IOERROR
```

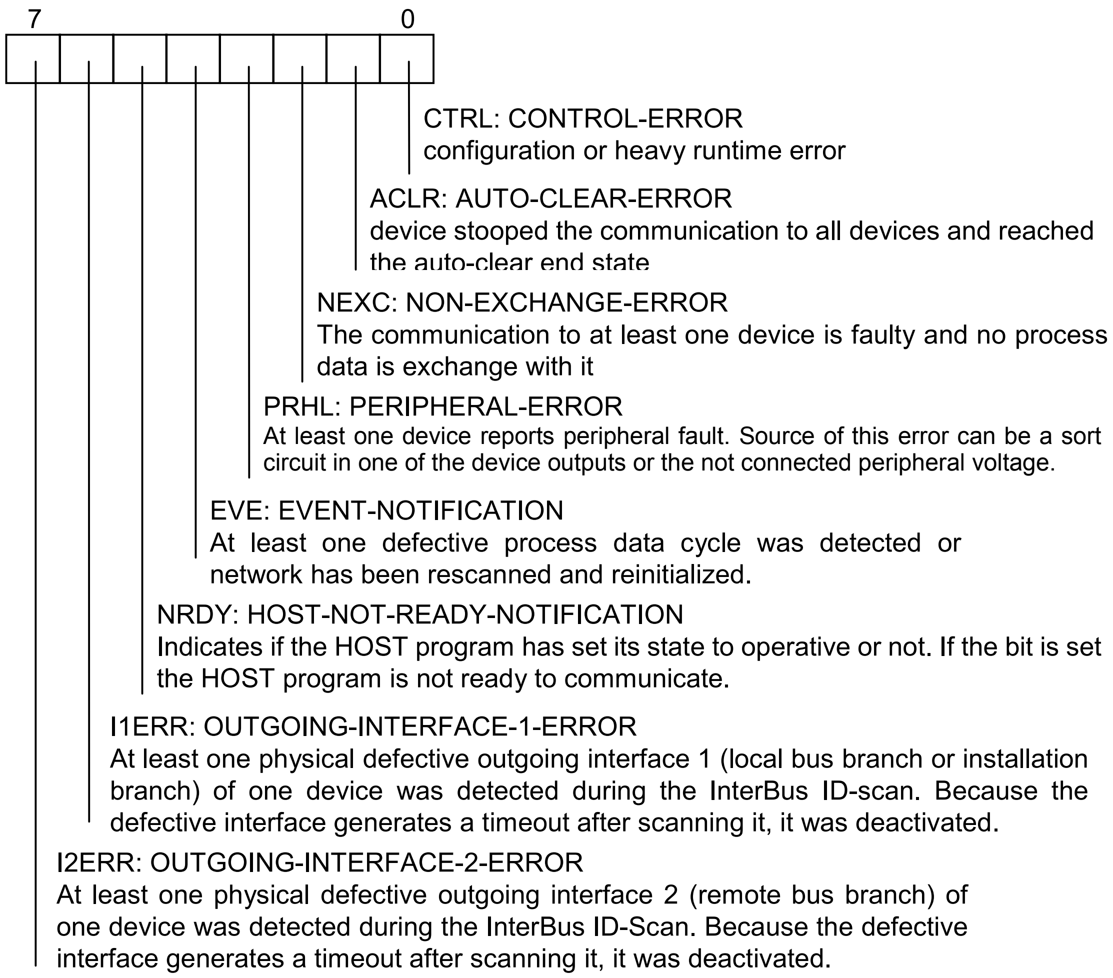
MD254

< COM-IBSM >


7
0
0
0
Ctrl: Configuration list differs from the actual connection
W1Err: Defective slave in level 1 after initialization
W2Err: Defective slave in level 2 after initialization
SysErr: Bus communication error
Mod: Slave error
Rec: Reconstruction request from slave

- GlobalFlags (1byte) that it is informed from a field-bus module is stored.
- When an error occurs, either Ctrl or W1Err, W2Err, SysErr turns it on. At this time, the details code of the error is stored in MD255. It is 0 at the time of the normality.

< COM-IBM >



CTRL: CONTROL-ERROR
configuration or heavy runtime error

ACLR: AUTO-CLEAR-ERROR
device stooped the communication to all devices and reached the auto-clear end state

NEXC: NON-EXCHANGE-ERROR
The communication to at least one device is faulty and no process data is exchange with it

PRHL: PERIPHERAL-ERROR
At least one device reports peripheral fault. Source of this error can be a sort circuit in one of the device outputs or the not connected peripheral voltage.

EVE: EVENT-NOTIFICATION
At least one defective process data cycle was detected or network has been rescanned and reinitialized.

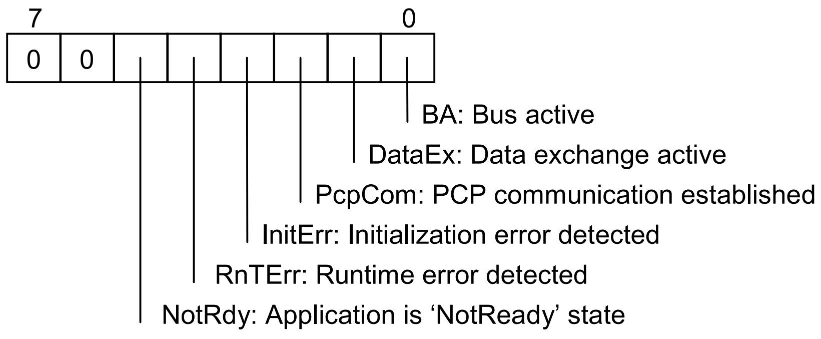
NRDY: HOST-NOT-READY-NOTIFICATION
Indicates if the HOST program has set its state to operative or not. If the bit is set the HOST program is not ready to communicate.

I1ERR: OUTGOING-INTERFACE-1-ERROR
At least one physical defective outgoing interface 1 (local bus branch or installation branch) of one device was detected during the InterBus ID-scan. Because the defective interface generates a timeout after scanning it, it was deactivated.

I2ERR: OUTGOING-INTERFACE-2-ERROR
At least one physical defective outgoing interface 2 (remote bus branch) of one device was detected during the InterBus ID-Scan. Because the defective interface generates a timeout after scanning it, it was deactivated.

• GlobalBits (1byte) that it is informed from a field-bus module is stored.
• All bits are error information. The details code of the error is stored in MD255. It is 0 at the time of the normality.

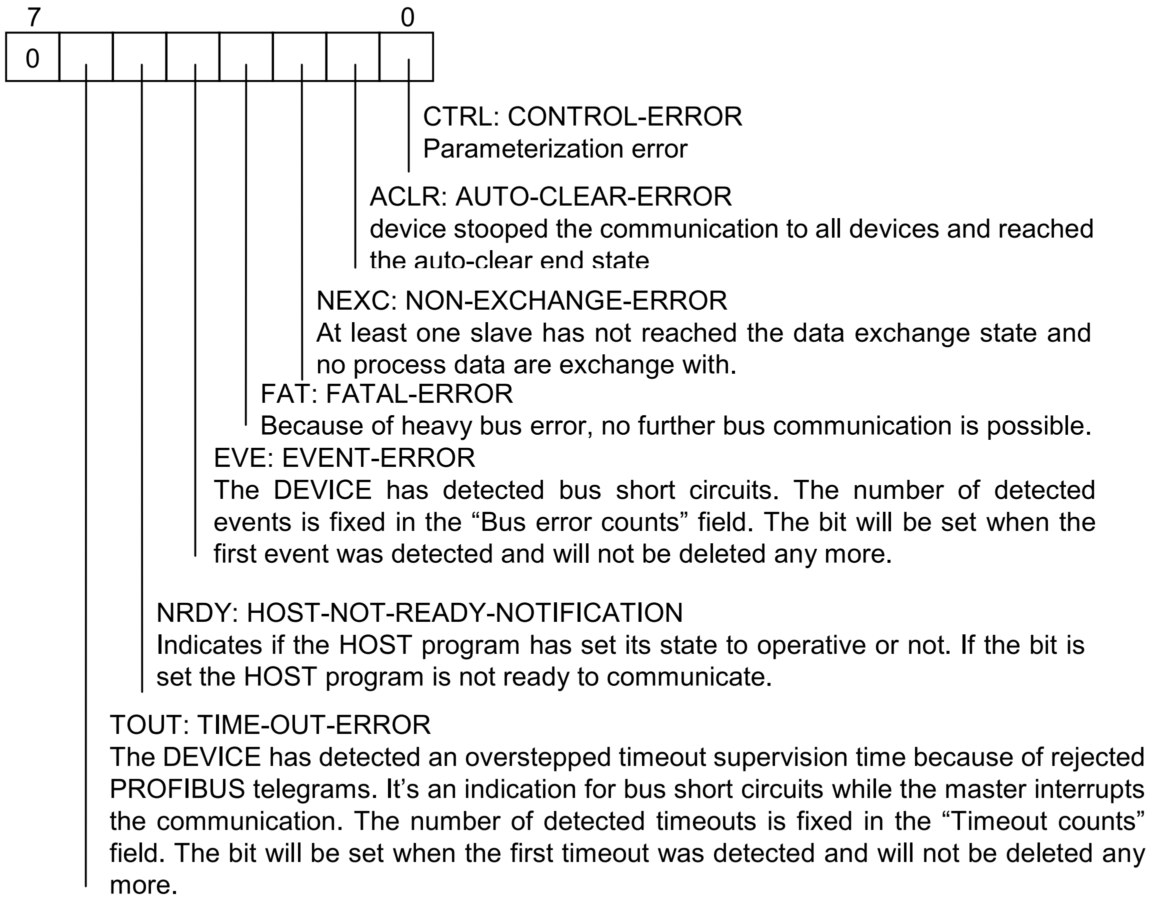
< COM-IBS >



BA: Bus active
DataEx: Data exchange active
PcpCom: PCP communication established
InitErr: Initialization error detected
RnTErr: Runtime error detected
NotRdy: Application is ‘NotReady’ state

• GlobalBits (1byte) that it is informed from a field-bus module is stored.
• RnTErr turns it on when an error occurs during the running. The details code of the error is stored in MD255 at this time.

< COM-PB, COM-FMS, COM-DPM >



CTRL: CONTROL-ERROR
Parameterization error

ACLR: AUTO-CLEAR-ERROR
device stooped the communication to all devices and reached the auto-clear end state

NEXC: NON-EXCHANGE-ERROR
At least one slave has not reached the data exchange state and no process data are exchange with.

FAT: FATAL-ERROR
Because of heavy bus error, no further bus communication is possible.

EVE: EVENT-ERROR
The DEVICE has detected bus short circuits. The number of detected events is fixed in the “Bus error counts” field. The bit will be set when the first event was detected and will not be deleted any more.

NRDY: HOST-NOT-READY-NOTIFICATION
Indicates if the HOST program has set its state to operative or not. If the bit is set the HOST program is not ready to communicate.

TOUT: TIME-OUT-ERROR
The DEVICE has detected an overstepped timeout supervision time because of rejected PROFIBUS telegrams. It’s an indication for bus short circuits while the master interrupts the communication. The number of detected timeouts is fixed in the “Timeout counts” field. The bit will be set when the first timeout was detected and will not be deleted any more.

• GlobalBits (1byte) that it is informed from a field-bus module is stored.
• All bits are error information. The details code of the error is stored in MD255. It is 0 at the time of the normality.

< COM-DPS >

• It is completely unavailable information. A zero is always stored.

MD255

• An error details code (01H～FFH) is stored at the time of the error occurrence.
• LastError (1byte) that it is informed from a field-bus module is always stored in case of using COM-DPS. ErrorEvent (1byte) is stored in case of using the one except for it.
• At the time of the normality, 0 stored.
• Refer to the interface manual of each field-bus module for the details of the error code.

MW254

- The number of free messages of the field-bus message communication buffer inside STC OS is always stored.
- A field bus message communication buffer can store a maximum 64 message. And, 1 message is used for this buffer by writing to the network variable or the execution of the NetWrite function.
- It has the possibility that it becomes buffer-full when the execution of a substitution to the network variable or a NetWrite function has a short cycle. STC OS abandons transmitting data under the condition of buffer-full, and it isn't made an error specially.
- It is possible that buffer-full is watched with the application program like the following.

```
        DIMNET SEND%
*LOOP
        IF MW254 = 0 THEN           ‘Buffer full
                DELAY 0.5           ‘500 msec Wait
                GOTO *LOOP
        ENDIF
        SEND% = 1                   ‘Write to the network variable
```

MW255

- The error code which STC OS detects inside is stored. It is 0 at the time of the normality.
- As for the details of the error code, it is the following.
    - Error code notified from field-bus module. It has the same code as MD255.

    1 to 255: Refer detail of error code to interface manual of each field-bus module.
    - Error code that STC OS raises

    1001 (03E9h): Indication received when waiting for confirmation
    1002 (03EAh): Invalid CR number in the received confirmation
    1003 (03EBh): Invalid message ID in the received confirmation
    1004 (03ECh): Receiving time-out for confirmation
    2001 (07D1h): Confirmation received when idle

2.3 TIMER

HARL-III offers 4 types of timers.

(1) TIMm ....Universal timer or watch dog timer

1) TIMm = <value>

The timer starts upon executing LET command. The counting timer is not interrupted by JOB OFF command.

2) "m" means timer number. Specify the number from 0 through 31.

3) The valid value is from 0.00 through 327.67 seconds. (The value 0 is converted to - 1 immediately.)

Note: In HARL-III, the value "- 1" is regarded as true. However, there is no value "-1" in timer. In order to describe the time up condition as true, the value is converted to - 1 when the value becomes 0. Therefore, in case of example 2, the timer can not work properly. Express as example 1 below.

4) Any timer can be used at each job. However, if a new value is set to a timer which is counting down, the value of the timer is rewritten to the new value.

5) The condition of time Up can be used by IF sentence. In this case, time up condition is true.

Example 1) .... recommended usage

IF TIM3 THEN GOTO * TIMEUP (After TIM3 is time up, jump to * TIMEUP.)

Example 2) .... not recommended usage

IF TIM3 = 0 THEN GOTO * TIMEUP (Because there is not instant at time 0, the program is not executed correctly. Describes the program as Example 1) above.)

(2) DELAY command .... Pause of program execution

After DELAY command, the program does not step to next until the time specified by DELAY command passes by.

(3) Internal calendar clock

Set and read the time or date by using TIME $ or DATE $ command.

(4) WAIT timer with and without TIMEOUT variable

While waiting to satisfy the condition set by WAIT command, the timer for the wait condition can be set by TIMEOUT variable.

2.4 FILE and Communication

In HARL-III the communication with external devices is executed with "FILE" concept. For example, the communication or instruction execution to a robot is processed as the file handling.

Communication port connected to a robot is assigned as a file with a number (# n) by OPEN "COM command. (See section OPEN COM command in chapter 4.)

After executing OPEN COM command, the file or the robot is specified by the file number (#)

Note: When using the file,

(1) Once a file is opened by a job, it can not be opened again in another job until it is

(2) A communication port can not be opened as a different one with a different number at the same time.

The followings can be used as communication port.

Type	**Port number**	**Explanation**
Dual port RAM communication port	0	This is used for high speed communication with a robot controller. The controller connected to a robot through dual port RAM can be used.
RS-232C communication port	1	Normal communication port HNC robot controller can be connected.
Expansion communication port	2 3	Normal communication port. HNC robot controller or other intelligent equipment can be connected.
Host communication port	8	Communication port for host computer or HNC robot controller can be connected.
Programming console port	(9)	Can not be specified in a user program.

Table 2.8

Note:

If OPEN command is executed to the HOST port (COM8), the communication by the protocol, HRCS-VI, is prohibited, but the port is used to connect HNC robot controller.

It is recommended that communication through a serial port is handed only in one Job to avoid communication error. Exceptions are commands like DISABLE and REF.

2.5 Error Processing

If some error happens during the execution of a program, some error processing is required in order to avoid program execution from being terminated.

In order to perform the error processing, define the error branch by ON ERROR GOTO command at the top of a job and create some error processing routine(s) as sub-program(s).

In an error processing sub-program, the type of error may be identified by ERR command, RESUME command is required at the end of the sub-program to return to the main program.

3. HARL-III LANGUAGE RULES

3.1 Sentence

Sentence is the smallest program unit of program. A sentence consists of one command. A command consist of command name and data (operand and operator). Operand can be specified constant, variable number or function.

Example)

$$\underbrace{\underbrace{\text{DELAY}}_{\text{Command name}}\quad\underbrace{\text{5.2}}_{\text{Operand}}}_{\text{Command (Sentence)}}$$

3.2 Line

Generally a line consists of one sentence. A sentence can be written 254 characters in one line. Multiple sentences can be written in one line by dividing each sentence with colon (:). This is called multi-statement and several sentences can be written in one line.

3.3 Line Number

HARL-III does not require any line numbers.

3.4 Character Set

The characters which can be used are as follows.
The uppercase and lowercase letters of alphabet, numbers, special symbols.

3.5 Special Symbols

There are some special symbols in addition to the arithmetic operators (+, -, *, /) below.

(1) Colon(:)

It is used to separate the sentences of multi-statement as a terminator.

Example)

```
A=B+C:X=A
```

(2) Comma (,)

It is used to separate parameters.

Example)

MOVE #1, PTP, PM0

(3) Semicolon (;)

It is used to separate PRINT sentence.

Example)

PRINT #1, " A = "; A

(4) Apostrophe (')

It can be used instead of REM sentence (comment sentence). Any letters after an apostrophe in a line are ignored as a comment.

(5) Asterisk (*)

When defining labels at the beginning of line, the label names must follow after " * ".

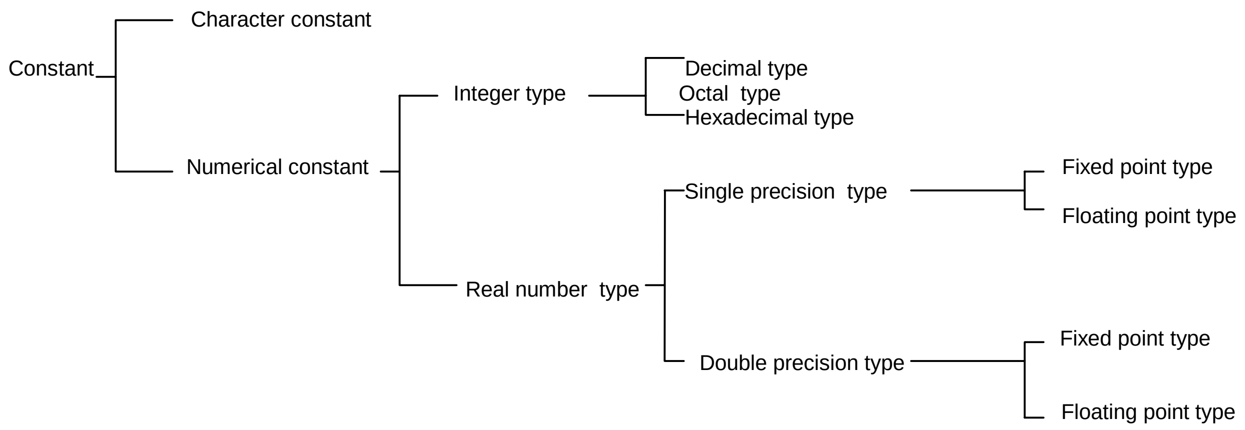
(6) space

It is used for a blank of characters or for break point which is specified as format.

3.6 Constant

3.6.1 Constant Type

The available constants by HARL-III are as follows.



3.6.2 Character Constant

Character constant is the string expressed by alphabet, numbers, katakana characters within 255 characters enclosed by double quotation marks ("). The double quotation marks can not be expressed directly in the character constant. If double quotation marks are required in the character constant, express them as shown the example below.

Example)

(1) "Good morning"

(2) " 12343567879"

(3) CHR $ (& H22) + " 1234"+ CHR $ (&H22)

3.6.3 Numerical Constant

Numerical constants can be divided into integer type and real number type and each type has positive, negative or 0. Minus symbol must be put the top of negative numbers, but plus symbol can be omitted in case of positive numbers.

3.6.4 Integer Type

(1) Decimal type

All integer type from - 32768 through + 32767 or numbers with percent mark (%) are available in case of decimal type. Decimal point can not be used.

Example)
```
..................  32767
......................- 123
................... 32767 %....  . means integer
```

(2) Octal type

The octal type is the numerical string which consists of the numbers from 0 through 7 with & or &O mark at the top of number. The applicable range is from &0 through &177777.

Example)
```
        &12345
        &07777
```

(3) Hexadecimal type

The hexadecimal type is expressed by the character string from 0 through F with &H mark at the top of the string.

Example)

```
&H100
&HFFFF
```

Note:

In PRINT command, the values expressed by octal or hexadecimal number are output by decimal number.

3.6.5 Real Number Type

Real number type is divided into single precision type and double precision type.

3.6.6 Single Precision Type

The available digits of the single precision type is 7. The 7th digit is rounded to the nearest whole number and the value is displayed under 6 digits. The applicable range is from -1.70141E+38 through 1.70141E+38.

(1) Real number under 7 digits

(2) Exponent type with E

(3) The number with ! mark at the end of number

Example)

```
1.23
- 7.09E-06
3525.68
3.14!
```

3.6.7 Double Precision Type

The available digits of the double precision type is 16 and the value is displayed under 16 digits.

(1) The number more than 7 digits

(2) Exponent type with E

(3) The number with # mark at the end of number

Example)

```
1234567890
- 1.09432E - 06       + 0.3141592653E + 01
56789.0 #
8657036.1543976
```

3.7 Variable Number

(1) Variable number

The variable number consists of alphanumeric character which are corresponded to the area to store the value used in program. The value of variable number is defined by program and can be used for operation and reference. The value of variables is undefined right after download of program. Even if the power of the controller is reset, the variables are not initialised.

(2) Variable name and type declaration character

All variable names are expressed by alphanumeric started with alphabet (Max 16 words) and a period (.).

Example)
Each of two variables below is handled as different variable name.

A.1234560
A.1234568

The variable name must not be a reserved word, but the variable name with a reserved word is available. The variable name started with FN is not available. When using alphabet, there is no difference between upper-case and lower-case letters. The same variable names can be used in different variable type. The variable type is defined by type declaration character which is put at the end of variable number. The variable number without type declaration character is the same as with " ! " mark and recognised as single precision type.

Type declaration character	% Integer type
	! Single precision type
	# Double precision type
	$ Character type

Example)

A
A #   Each of them are different each other. But A and A! is the same type.
A %
A $

(3) Array variable

An array is a variable number which can refer some elements. Each of elements is referred by integer or subscripts expressed by integer. Dimension range of array variable is from 1 through 3 and the subscripts are allowed to use within memory range defined by DIM sentence. Each subscript starts from 0.

Example)

DIM A (10) 1 dimension array, amount of elements = 10

DIM TA (10,50) 2 dimension array, amount of elements 10 x 50 = 500

DIM TTA$ (2,5,3) 3 dimension array, amount of elements 2 x 5 x 3 =30

Note: When the amount of each subscript is under 10, DIM sentence can be omitted.

(4) Reserved variable

The variables shown below can not be used as general variable by user.

TIME $	:This variable has current time as HH:MM:SS in sequence of hour, minute and second.
DATE $	:This variable has current date as YY:MM:DD in the sequence of year, month and date.
ERR	: This variable has current code when the error occurs. It can not be substituted.
TIMEOUT	:When using WAIT command with the time limitation, the result, whether or not the time was over, is stored.

See also appendix reserved words which cannot be used also.

(5) Memory

There are lots of memories as shown in section 2.3 and each is assigned the function, bit numbers and memory numbers. When the memory is specified, express the memory name and number in order. The numbers can be specified directly or through memory pointer using other memories or variables.

Example)

(1) INB10	Direct designation
(2) INB(MD5)	Memory pointer or indirect designation

Use parentheses in case of indirect designation or memory pointer. When a memory is specified in parentheses, the memory can be also specified indirectly. Indirect designation using expression is possible. Even if the power of the controller is reset, the contents of memories are not initialised.

3.8 Local Variable, Global Variable and Network Global Variable

In HARL-III the variable used in one job with variable names are called local variables. (There is no command to define the local variable.) Local variable is independent of each job. The same variable names can be used in other job. However, the local variables in a job can not be referred and/or changed by other jobs. Therefore, modularization and re-using of program can be easy.

The variables used in all jobs are called global variables and the can be referred and changed from other jobs. The data can be sent and got between jobs by using global variable. Memory data such as MB or MD is used as global variable unconditionally (It is also recognized as global variable after changing the name by DEFINE statement). Among reserved variables, TIME$ and DATE$ are global variable and ERR is local variable.

The variables declared by DIMNET statement are called network global variables. These variables can be read or written by all STC in the network.

The available local variables are 200 per one job and the available variables are 300 in total at maximum.

3.9 Type Conversion

Numeric data can be converted to other type. However, the conversion between numeric type and character type is not possible.

(1) When the numeric data of some type is substituted for numeric variable of other type, the value is converted to the type declared by its variable name.

Example)
ABC% = 1.234 (1 is substituted for ABC)

(2) In case of the operation between different precision, the value is converted to higher precision at operation. 10 # /3 is operated as 10 # /3 #.

Example)
A # = 10 # /3 (3.33333333333333333333 is substituted for A #)
B # = 10 # /3 # (3.33333333333333333333 is substituted for B #)

(3) In case of logic operation, all numerics are converted to integers and the result are shown by integers.

Example)
A = 12.34 (12.34 is substituted for A)
B = NOT A (- 13 is substituted for B)

(4) In case of conversion from real number to integer, the value under decimal point is rounded to the nearest whole number. In this case, if the value is over the integer type, the error is indicated .

Example)

```
A % = 34.4                    (34 is substituted for A)
B % = 34.5                    (35 is substituted for B)
A#= 1.234E+0.7
B %=A#                        (Overflow at the line B % = A # )
```

(5) When the double precision variable is substituted for the single precision variable, the value is expressed as significant 7 columns. The precision variable is 7digits and the absolute value of the error against the original value is less than 5.96E - 8.

Example)

```
A # = 1.23456789 #            (1.23456789 is substituted for A # )
B ! = A #                     (1.234567 is substituted for B ! )
```

Note: When the operation is mixed with the double precision variable (or constant) and the single precision variable (or constant) or the value of the single precision is substituted for the double precision variable, the conversion error happens at the digits after significant columns.

Example)

(1) The operation between different precision (Conversion error happens in the operation result)

```
Bad example        : A # = 1.41421356 # + 0.12
Good example       : A # = 1.41421356 # + 0.12 #
```

(2) When lower precision value is substituted for higher precision;

```
Bad example        A # = 3.1415
Good example       A # = 3.1415 #
```

3.10 Expression and Operation

Expression is the general numeric expression as constants and variables connected with operator. Also, the characters and numerics or any variables are regarded as expression.

Example)

```
"BASIC"
3.14
10+3/5
A + B/C - D
TAN (DO)
```

HARL-III operation is sorted into 5.

1) Arithmetic operation

2) Relational operation

3) Logical operation

4) Function

5) Character string operation

3.10.1 Arithmetic Operation

Arithmetic operation are shown as below.

	Operator	**Operation**	**Example**
Priority of operation ↓	^	Exponent operation	X^Y
	-	Minus sign	-X
	*, /	Multiplication, division	X*Y, X/Y
	+, -	Addition, subtraction	X+Y, X-Y

In case of changing priority, use parentheses. Operator enclosed by parentheses is processed earlier than other operation. In parentheses, operation is executed in the sequence.

The following shows the execution example.

	Arithmetic operation	**HARL-III expression**
1)	$2X+Y$	2*X+Y
2)	$\frac{X+2}{y}$	X/Y+2
3)	$\frac{X+Y}{2}$	(X+Y)/2
4)	$X^2 + 2X + 1$	X^2+2*X+ 1
5)	X^{Y2}	X^(Y^2)
6)	$(X^y)^2$	X^Y ^2
7)	$Y(-X)$	Y * -X

(1) Multiplication and division of integer

Operation of multiplication or division is processed by MOD. In case of real number, the value below the decimal point is rounded to the nearest whole number before operation.

Example)

13.3 MOD 4 = 1 (13/4 = 3 ... 1)

25.68 MOD 6.99 = 5 (26/7 = 3 ... 5)

(2) Division of Zero

When the expression is divided by 0, the maximum number processed internally is substituted as quotient and it is processed as error. In case of minus exponentiation to 0, the process is the same as the division by 0.

Example)

```
A = 2/0                  A = 0^ - 1

1.70141E + 38          1.70141E + 38
```

(3) Overflow

When the result of operation or substitution is over the allowed range, overflow occurs. When the overflow happens, overflow error is output and the maximum number is given as result and it is processed as error.

Example)

```
          A = 3^300
          1.70141E + 38
```

(4) Exponent

Exponent operation can not be calculated by negative real number. (positive number or negative integer is possible.)

3.10.2 Relational Operation

Relational operation is to compare two numerics. The result is given by true (-1) or falsehood (0) and used to branch the program flow conditionally. (See IF command)

Relational operator	Description	Example
	Equal	X=Y
<>,><	Not equal	X<>Y,X><Y
<	Smaller	X< Y
>	Larger	X > Y
<=, =<	Small or equal	X <= Y, X =< Y
>=, =>	Large or equal	X >= Y, X => Y

Note: = (equal) is also used for assignment statement.

The followings are the samples in IF command.

```
IF X = 0 THEN GOTO *GO1

IF A+B<>O THEN X=X+ 1
```

3.10.3 Logical Operation

Logical operation is used for checking several conditions, bit operation or pool operation. Logical operation gives 0 or 1 to each bit as result.

Descriptions of each logical operator are shown below.

NOT = not (Negation)

X	NOT X
1	0
0	1

AND = and (Logical multiplication)

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

0R = inclusive or (logical addition)

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

X0R = exclusive or (exclusive logical addition)

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

Logical operation is also used for changing program flow as relational operator. In this case, logical operator can be connected by one relational operator and more.

Example)

```
(1)   IF X < 0 OR 99 < X THEN GOTO *GO1
(2)   IF 0<XANDX< 100THENX=0
(3)   IF NOT (A = 0) THEN GOTO *GO2
```

(1) If X is larger than 99, jump the program to the label *GO1.

(2) If X is plus and smaller than 100, 0 is substituted for X.

(3) If A is not 0, jump the program to the label *GO2.

Before operation, logic operator converts the numerics to integer expressed by complement from - 32768 through + 32767. If the result is beyond the range, overflow occurs. When only0 (falsehood) or -1 (true) is given, logic operator gives 0 or -1 as result. In specified logical operation, it is operated in bit unit. Therefore, logical operator can check the byte data by referring bit pattern. For example, AND operator can be used for masking one byte input data (IND) for the bit to be checked. OR operator can mix two bit patterns for making a binary number.

Example of logical operator function.

63 AND 8 = 8	63=(111111)2,8=(001000)2 Therefore, 63 AND 8 = (001000)2= 8
-1 AND 8 = 8	-1 = (1111111111111111)2, 8 = (001000)2 Therefore, -1 AND 8 = 8
12 OR 11 = 15	12 = (1100)2, 11 = (1011) 2 Therefore, 12 OR 11 = (1111)2= 15
32767 0R -32768 = - 1	32767=(0111111111111111)2 -32768 = - (1000000000000()00) 2 Therefore,32767 0R-32678 = (1111111111111111)2= -1
12 XOR 11 = 7	12 = (1100)2, 11 = (1011)2 Therefore, 12 XOR 11 = (0111)2 = 7
10 XOR 10 = 0	10 = $(1010)_2$ Therefore, 10 XOR10 = (0000)2 = 0
- X = NOT X - 1	A minus number is complement of 2 adding 1 to its complement of 1.

3.10.4 Function

Function is a specified operation to a given argument.

There are numeric functions such as SIN (sine) or SQR (square root) and character string functions such as CHR $ or MID $ in HARL-III. In these functions, the double precision is applied when the argument is by double precision and the single is applied when the argument is by integer or single precision.

Example)

```
A = SIN (3.14) + COS (3.14)
PRINT# 1,2,2* 2,SQR(2)
```

3.11 Character String Operation

3.11.1 Connection of Character String

Character strings can be connected by using operator "+".

Example)

A $ = "HIRATA": B $ = "INDUSTRIAL": C $ = "ROBOT"
D$=A$ + " " + B$ + " " + C$
("HIRATA INDUSTRIAL ROBOT" is substituted for D $.)

3.11.2 Comparison of Character Strings

Characters are also compared by using the same relational operator as numeric values.

=, <, >, 0, >< ,<=, =<, >=, =>

In case of character string, each character is compared from the first character. When comparing two character strings, if there is a different character, the character string which has larger character code becomes larger.

Note: When comparing the character strings, a blank is also regarded as a character.

Example)

"AA" < "AB"

"BASIC" = "BASIC"

"X &" > "X #"

"PEN " > "PEN"

"cm" > "CM"

" DESK" < " DESKS "

As shown above, the comparison of character strings can be used to check the contents of character string or to sort the characters in the sequence of alphabet.

3.12 Priority of Operation

Operation is processed according to the priority as below.

1. Enclosed by parentheses

2. Functions

3. Exponentiation (^)

4. Minus (-)

5. * , /

6. MOD

7. +, -

8. Relational operator (<, >, =, etc)

9. NOT

10. AND

11. OR

12. XOR

3.13 Label

IF or GOTO command is to change the program flow.

Example)

```
OUTD0 = 0   ..........................................Turn off OUTD0 (OUTB0 ~ 7).
A = IND0 - IND1 ....................................Comparing IND0 and IND1
IF A < 0 THEN GOTO *MINUS  ...............When INDO < IND1, jump to *MINUS.
IF A > 0 THEN GOTO *PLUS  .................When INDO > IND1, jump to *PLUS.
OUTB0 = 1  ............................................Turn on OUTB0 (IND0 = IND1).
GOTO *EXIT ...........................................Jump to *EXIT.
*PLUS: OUTB 1 = 1 ...............................Turn on OUTB 1 (IND0 > IND 1).
GOTO *EXIT............................................Jump to *EXIT.
*MINUS: OUTB2 = 1 ..............................Turn on OUTB2 (IND0 < IND1).
*EXIT: OUTB7 = 1 ..................................Turn on OUTB7 (END).
```

Label name is a mark of a program line to be jumped conditionally or unconditionally.

Note:
1. Put the asterisk mark (*) at the top of label name.

2. Label name must be started with alphabet except the asterisk.

3. The characters which can be used as the label name are alphanumeric characters and period (.) except asterisk. There are no difference between uppercase and lowercase letters.

4. The reserved words (MOVE etc) can not be used as the label. However, the label name including the reserved word is possible.

5. Label name must be within 8 characters.

6. Label name to be referred must be set at the top of program line.

7. In case of multi statement inputting a command after the label name in the same line, divide the label with a sentence by a colon (:) or a space.

If the above notes are ignored, "Syntax error" occurs.

The same label name can be used at different jobs. If the same label names exist in one job, "Duplicate label " error occurs.

The maximum labels available per one job are 1500.

4. COMMANDS OF HARL-III

4.1 List of HARL-III Commands

Sort	Use	Command	Function
Psuedo-instruction	Definition	JOB NAME	Set the first job and job name.
Definable instruction	Definition	DIM	Define as array variable.
		DIMNET	Define as network global variable
		GLOBAL	Define as global variable.
		DIMPOS	Define the number of position memory.
		REM	Define the comment line.
General instruction	Substitution	LET	Substitute value to variable.
		PULSE	Substitute value for specific time.
	Flow control	GOTO	Jump to a specified line, then execute.
		GOSUB	Call subroutine.
		RETURN	Terminate subroutine, then resume the former process.
		FOR TO STEP~NEXT	Repeat the instruction between FOR and NEXT.
		IF THEN ELSE	Decide the condition of logical expression.
		DELAY	Break temporarily the execution of job.
		WAIT	Wait until conditions are satisfied.
		TIMEOUT	Get the result of timeout by WAIT command.
		ON GOTO/ON GOSUB	Branch to a specified line.
		SELECT CASE	Evaluate an expression and execute the processing block.
Interrupt control instruction	Error control	ON ERROR GOTO	Specify the destination at error.
		RESUME	Terminate error process, then resume the former process.
		ERR	Hold error code.

Sort	Use	Command	Function
Control instruction	Job control	JOB START JOB ON JOB OFF	Control job execution.
	Robot control	MOVE	Move a robot to specified coordinates.
		SET	Set operating characteristic data of a robot.
		REF	Deal data inside of a robot.
		SEQ~SEQEND	Set or terminate robot sequence mode.
		FINISH	Complete MOVE in sequence mode.
		HOLD	Specify or cancel the servo lock of the robot.
		DISABLE	Prohibit robot movement.
		CALIB	Execute automatic origin calibration.
		SETROBNO	Set a robot number for the robot communication.
		CLEARROBNO	Clear a robot number for the robot communication.
		GETROBNO	Get a robot number for the robot communication.
	File control	OPEN “COM~	Open a communication file.
		CLOSE	Close a file.
		INPUT$	Read the specified lemgth of the character strings from a specified file.
		INPUT #	Substitute data of a sequential file to a variable.
		LINE INPUT #	Read one line from a sequential file.
		PRINT #	Output data to a file.
		EOF	Examine the termination code of a file.
	Clock control	TIME$	Get time. Setting is possible.
		DATE$	Get date. Setting is possible.
Network instruction	Network communication	NETOPEN	Open a network communication.
		NETCLOSE	Close a network communication.
		NETREAD	Read data from a network communication.
		NETWRITE	Write data from a network communication.

Sort	Use	Command	Function
Conversion instruction	Arithmetic function	SIN	Get sine.
		COS	Get cosine.
		TAN	Get tangent.
		ATN	Get arctangent.
		SGN	Get the sign of value
		ABS	Get absolute value.
		INT	Remove decimals
		FIX	Remove decimals
		LOG	Get natural logarithms.
		EXP	Get e raised to a power.
		SQR	Get square root.
	Arithmetic Constant	PAI	Get the value of π.
	Character	LEFT$	Pick out arbitrary length from the left of character strings.
		MID$	Specify one part of character strings.
		RIGHT$	Pick out arbitrary length from the right of character string.
		SPACE$	Get arbitrary length blank character strings.
		CHR$	Get the character of specified character code.
		STRING$	Get the character strings connected one arbitrary character.
		HEX$	Get the character strings converted decimal into hexadecimal.
		STR$	Convert numerical value into character strings.
		VAL	Convert the number of character string display into actual value.
		ASC	Get the character codes of characters.
		LEN	Get the total byte count of character strings.
		INSTR	Get the position of the specified character strings in character strings.

4.2 Explanation of Each Command

4.2.1 How to Read This Section

All command of HARL-III are explained in this section. The explanation of each command is constructed as below.

[Function]

[Format]

[Example]

[Explanation]

[Function] ...... Explains the function of command.

[Format] .... Explains the format using the command. At the entry of command, follow the .... rules below.

(1) When typing in the commands, there are no difference between the uppercase and lowercase letters. However, in case of the character enclosed by double quotation marks (") except for the file name, distinguish between the uppercase and lowercase letters of alphabet.

(2) When the space (_ underline) is specified, enter a blank as one character.

(3) User must specify the items enclosed by angle brackets "< >".

(4) The items enclosed by square brackets "l[ ]" are optional and can be omitted. When omitting the bracket, the default value (the value which has already set in HARL-III) or the value specified before is applied.

(5) The symbols, except for "< >" and "[ ]" above, parentheses "( )", comma (","), semicolon (";"), minus symbol and equal symbol ("=") etc must be typed in the specified place.

(6) The item which has ellipsis (...) can be repeated within the allowable length of one line. (255 characters at the maximum)

Example) <constant> [,<constant>...] In this case 0, 10, 15

[Example] ....Shows an example for the entry of format.

[Explanation] .....Explains the details of function, notice and usage of command.

ABS (function)

[Function]

Gets the absolute value.

[Format]

ABS(<Numeric expression>)

[Example]

B=ABS(- 2)

The absolute value of - 2 (= 2) is substituted for the variable B.

[Explanation]

The absolute value of <Numeric expression> is returned. When <Numeric expression> includes double precision real number, the value is returned by double precision. In other case, the value is returned by single precision.

ACCEL (motion parameter memory)

[Function]

This value influence the acceleration and deceleration of the robot motion

[Format]

ACCEL=<Value>
ACCEL=([<Value of 1st axis>], [<value of 2nd axis>],[...])

[Example]

SET #1,ACCEL=80
SET #1,ACCEL=80,50,20

[Explanation]

When the power of the controller is turned on the values of ACCEL which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group).
If later a SET command is executed by the HARL program the value will be overwritten.
<Value> or <Value or N axis> is specified by the percentage of maximum acceleration speed.
(<Value>: 0 to 100) This is used in PTP, GATE, ARCH PASS motion.

Refer **SET**

AND (logical Operator)

[Function]

Logical operation is used for checking several conditions, bit operation or pool operation. Logical operation gives 0 or 1 to each bit as result. AND = and is a logical multiplication.

[Format]

< numerical expression>_ AND_<numerical expression>

[Example]

1) IF 0<X AND X< 100 THEN X=0
   If X is plus and smaller than 100, 0 is substituted for X.

2) 63 AND 8 = 8
   63=(111111)2,8=(001000)2
   Therefore, 63 AND 8 = (001000)2= 8

   -1 AND 8 = 8
   -1 = (1111111111111111)2, 8 = (001000)2
   Therefore, -1 AND 8 = 8

[Explanation]

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

Logical operation is also used for changing program flow as relational operator. In this case, logical operator can be connected by one relational operator and more.

Before operation, logic operator converts the numerics to integer expressed by complement from - 32768 through + 32767. If the result is beyond the range, overflow occurs. When only 0 (falsehood) or -1 (true) is given, logic operator gives 0 or -1 as result. In specified logical operation, it is operated in bit unit. Therefore, logical operator can check the byte data by referring bit pattern. AND operator can be used for masking one byte input data (IND) for the bit to be checked

Refer **OR, XOR**

ARCH (motion parameter memory)

[Function]

This value influence the arch motion of robot motion path between vertical and horizontal axes.

[Format]

ARCH=<Value1>,<Value2>

[Example]

SET #1,ARCH=20,30

[Explanation]

When the power of the controller is turned on the value of ARCH UP and ARCH DOWN which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group). If later a SET command is executed by the HARL program the values will be overwritten.
<Value 1>: Starting position of Arch up motion.
<Value 2>: End position of Arch down motion.

Note :

The value of ARCH motion is related to the Z=0 and PULL UP. If value of ARCH is smaller than PULL UP no arch motion is performed.

Refer **SET, PULLUP**

<u>ARM (reserved variable)</u>

[Function]

This parameter is a component of memory P which defines the position of first and second arm of a SCARA robot.

[Format]

ARMm=<value>   < value > RIGHTY or LEFTY

[Example]

ARM2=RIGHTY

[Explanation]

The ARM component can be defined directly by the ARM command or by the REF command when an entire content of position memory PM.
< Value> can be RIGHTY or LEFTY. m is the number of the position memory

Refer **P, PM**

ASC (function)

[Function]

Gets the character code.

[Format]

ASC(<Character string>)

[Example]

A=ASC("A")

The character code of "A" (65) is substituted for the variable A.

[Explanation]

The ASCII code for the first character of <Character string> is returned.

Refer to **CHR$.**

ATN (function)

[Function]

Gets the value of arc tangent.

[Format]

ATN(<Numeric expression>)

[Example]

A=ATN(Y/X)

The arc tangent value of Y/X is substituted for the variable A.

Explanation]

The ATN function returns the angle whose tangent is <Numeric expression>. The range of the value is returned from - ~/2 through ~T/2 in radians. When <Numeric expression> includes double precision real number, the value is returned by double precision. In other case, the value is returned by single precision.

Refer to **COS, SIN, TAN**.

CALIB (command)

[Function]

Executes the automatic origin calibration (A-CAL) of robot.

[Format]

CALIB_#<File number>

[Example]

CALIB #1

[Explanation]

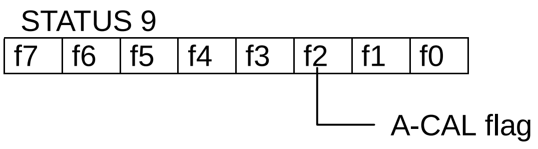
Starts the automatic origin calibration (A-CAL) of the robot which is connected to the communication port specified by <File number>. <File number> is specified by one of the number from 0 through 15. The communication port (0, 1,2,3 or 8) specified by <File number> had to be set by OPEN command.

After supplying the power to the robot controller, the A-CAL must be performed to coincide the origin of the robot body with that of its controller. If an error occurs during the A-CAL sequence the ON ERROR flag is set and the command will be interrupted. During the A-CAL the program remains at the program line.
The A-CAL can be also performed with the Teach Pendant.

The A-CAL status of the specified robot can be confirmed by the A-CAL flag in STATUS. Refer to section 2.2.7 Status.

A-CAL flag

The A-CAL flag is assigned in STATUS 9



IF (REF(#1, STATUS9) AND &H4) =0 THEN CALIB #1

The program above means, if the A-CAL of the robot connected to port 1 hasn't completed to perform CALIB command.

Refer to **OPEN, STATUS, REF**

CLEARROBNO (Function)

[Function]

Clears a robot number for the robot communication of a current job. (Sets the state without a robot number.)

[Format]

CLEARROBNO( )

[Example]

```
OPEN "COM1..." AS #1
OPEN "COM2..." AS #2
        .
SETROBNO(2)
MOVE #1,PTP,PM101   'COM1 robot number #2
        .
CLEARROBNO( )
MOVE #2,PTP,PM32    'COM2 without a robot number
```

[Explanation]

This function clears a robot number for the robot communication of a current job to set the state without a robot number.
After this function is executed, robot control commands such as **MOVE**, **REF** etc. communicates with a robot without a robot number.

After this function executed, the robot number got by **GETROBNO** function is –1.

[Errors]

(Compiling error)
- No error occurs.

(Execution error)
- No error occurs.

Refer to **SETROBNO**, **GETROBNO**.

Note) **CLEARROBNO** is available for the following version.

- STC
  Available ROM version is 5.41 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.01 or later.
  In case of older version, an error occurs when compiling.

CHR$ (function)

[Function]

Gets the character string of the specified character code.

[Format]

CHR$(<Numeric expression>)

[Example]

A$=CHR$(65)

The character "A" (65) is substituted for the variable A$.

[Explanation]

The value specified by the numeric expression is converted to the character code. Character code is listed in Appendix 1. If the value of <Numeric expression> is not from O through 255, "Illegal function call" error occurs.

Refer to **ASC.**

CLOSE (command)

[Function]

Closes the file.

[Format]

CLOSE_# <File number>

[Example]

CLOSE
CLOSE #1

[Explanation]

Closes the file opened by OPEN command. The closed file can not be input/output until the file is opened again. When <File number> is specified, the file corresponding to <File number> is closed. If <File number> is omitted, all opened files are closed. <File number> which was used to close the file can be used to open the same file or another file. The closed file can be opened again as the former or another <File number>.

Refer to **OPEN**.

COS (function)

[Function]

Gets the value of cosine.

[Format]

COS(<Numeric expression>)

Example]

C=COS(3. 1415/2)

The cosine value of 3.1415/2 is substituted for the variable C.

[Explanation]

<Numeric expression> is the angle whose cosine is to be calculated. The value of <Numeric expression> is in radians. If <Numeric expression> includes double precision real number, the value is calculated in double precision. In other case, the value is in single precision.

Refer to **ATN, SIN, TAN**.

DATE$ (reserved variable)

[Function]

Sets or retrieves the date of internal clock.

[Format]

1) DATE$

2) DATE$="yy/mm/dd"

[Example]

1) **B$ = DATE$** ....Retrieves the current date to B$.

2) **DATE$="99/01/01"** ..... Sets the date, January 1st, 99.

[Explanation]

The current date is always set in DATE$ as the form "yy/mm/dd" and can be retrieved anytime. There will be no problem with Y2K!

yy (Year)	00 to 99
mm (Month)	01 to 12
dd (Date)..............	0l to 31

When the time specified by TIME$ becomes "00:00:00", the date of DATE $ changes.

The value of DATE$ is handled as character string data, but can not be operated directly.

For example, when operating A$ = DATE$ + B$ should be expressed as;

```
D$ = DATE$
A$ = D$ + B$
```

Refer to **TIME$.**

DEFINE (command)

[Function]

This command is used in a header file and defines the relation between two expressions. Normally numerics will be substituted with alphanumerical expression for easier reading.

[Format]

DEFINE <Value1>_< Value2>

[Example]

DEFINE 12   Valve

[Explanation]

Many memories as MB, MD, MW, INB or OUTB are used in a program. In order to make an easier reading of the program the programmer can substitute the number of the memories by a alphanumerical expression. When the program will compiled compiler refer the header file and substitute the alphanumerical expression back to the numerical value.

Refer **INCLUDE**

DELAY (command)

[Function]

Pauses program (Job) processing.

[Format]

DELAY_<Numeric expression>

[Example]

```
DELAY 1.5
DELAY MW5/100
```

[Explanation]

The job processing is terminated after the execution of this command until the set time is passed. The value of <Numeric expression> must be set from 0.01 through 327.67 seconds.

The termination of the job execution is valid only for the job executed the DELAY command. The other jobs without the command are not affected the job executions.

If JOB OFF command is executed, the time count by DELAY command is paused. The count is continued by JOB ON command.

DIM (command)

[Function]

Specifies the maximum values for array variable subscripts and allocates storage accordingly.

[Format]

```
DIM_<Variable>(<Subscripts>[,<Subscripts>[,<Subscripts>]])
            [,<Variable>(<Subscripts>[,...]),...]
```

[Example]

DIM A (12,2), B $ (3)

[Explanation]

<Variable> must be filled the conditions of variable name. <Subscripts> should designate the maximum values for the array. The minimum value of subscript is 0. The number of array element is assigned as <Subscripts> +1. <Subscripts> can be specified up to 3 in one array variable and the array variable has the number of dimensions as specified. The numbers of the array elements or dimensions are limited by the size of memory.

Note: If a subscript is used that is greater than the maximum specified, "Subscript out of range" error occurs.

DIMNET (Declaration)

[Function]

Declares network global variables.

[Format]

DIMNET_<Variable>(<Subscripts>[,<Subscripts>[,<Subscripts>]])
[,<Variable>(<Subscripts>[,...]),...]

[Example]

```
DIMNET A%,B$,C!,D#,E%(10)
```

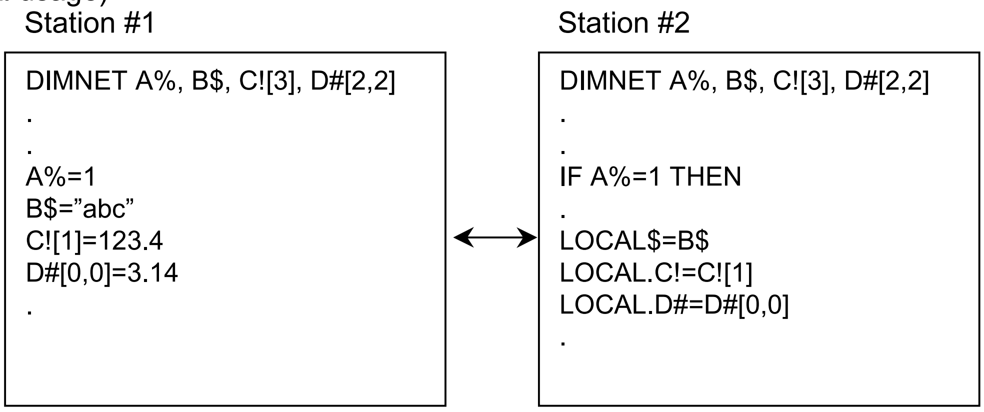
[Explanation]

When you use the network global variables in HARL-III program running in STC, it is necessary that you must create the network definition and download it to STC. Refer to operation manual of HARL-III Compiler about details.

How to specify the variables is the same as DIM statement. Refer to the explanation of DIM statement.

The variables declared by **DIMNET** have the shared memory in the network. They can be read or written by all STC in the network.

Typical usage)



```
DIMNET A%, B$, C![3], D#[2,2]
.
.
A%=1
B$="abc"
C![1]=123.4
D#[0,0]=3.14
.
```

```
DIMNET A%, B$, C![3], D#[2,2]
.
.
IF A%=1 THEN
.
LOCAL$=B$
LOCAL.C!=C![1]
LOCAL.D#=D#[0,0]
.
```

[Errors]

Errors are the same as **DIM**.

Refer to **DIM**.

Note) **DIMNET** is available for the following version.

- STC
  Available ROM version is 5.40 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.00 or later.
  In case of older version, an error occurs when compiling.

DIMPOS (Command)

[Function]

Specifies the numbers of the position memory "P".

[Format]

DIMPOS_<Numbers of P>

[Example]

DIMPOS 100

[Explanation]

When using the position data of the robot, the position memory "P" is applied. The numbers of P must be declared by <Numbers of P> according to the requirement. P is expressed with number such as P0, P1, P2. The example above shows that 100 positions, from P0 through P99 are declared. <Numbers of P> is specified by plus integer and the maximum number is 608 (P0 through 607). The content of a position memory is shown below. (See section 2.2.4.)

Pm							
PXm	Pym	PZm	PWm	ARMm	PDMm	PDF	S
X component	Y component	Z component	W component	ARM component	M component	F component	S code

"m" represents P number. When designating the number by expression or variable indirectly, express the number surrounding in parentheses.

Note:

1. The initial value of each component upon DIMPOS command is all 0s. Therefore, if P is attempted to substitute for PM, an error happens because of 0 (no set) to ARM component.

2. P is common position memory for all job. All jobs which use P must be declared by DIMPOS command in each job. If the numbers of P are different among the jobs, the largest number of P is applied as the memory area. However, the maximum number of P which can be used in a job is limited to the number declared in the job.

Refer **P, ARM, PX, PY, PZ, PW, PDM, PDF**

DISABLE (command)

[Function]

Forbids or interrupts the motion of the robot.

[Format]

```
DISABLE_#<File number>
```

[Example]

DISABLE #1

[Explanation]

Forbids the motion of the robot which is connected to the communication port specified by <File number>.

<File number> is specified by one of the value from 0 through 15. <File number> corresponded to the communication port (COM 0, 1, 2, 3 or 8) must be specified by OPEN command in advance.

If DISABLE command is executed while the robot is moving by MOVE command, the robot stops moving and the program step advances to the next.

The DISABLE status is cancelled by next MOVE command.

Refer **MOVE, OPEN**

EOF (function)

[Function]

Indicates an end of file condition.

[Format]

EOF(<File number>)

[Example]

IF EOF(1) THEN GOTO *ENDFI LE

If the file buffer of the file number 1 is empty, the program execution jumps to the line starting *ENDFILE.

[Explanation]

EOF is the function to check whether or not the file specified by < File number> reaches the end. EOF returns -1 (true) if end of file has been reached on the specified file. A 0 (false) is returned if end of file has not been reached. <File number> must be the same number specified by OPEN command. If the specified file has been opened by OPEN "COM command, the value of EOF function becomes true when the buffer is empty.

Refer **OPEN, INPUT**

<u>ERR (reserved variable)</u>

[Function]

Keep the error code when an error occurs.

[Format]

ERR

[Example]

IF ERR=7 THEN GOTO * ER7

If the error code is 7, the program execution jumps to the line starting *ER7.

[Explanation]

If the program line for the error processing has designated by ON ERROR GOTO command, the processing program according to the error code is executed on the occurrence of error. The value of ERR is cleared to 0 (no error) by RESUME or JOB START command.

Refer to **ON ERROR GOTO, RESUME, Error code list**.

EXP (function)

[Function]

Gets the value specifying e raised to a power.

[Format]

EXP(<Numerical expression>)

[Example]

```
X# = EXP(2.0)          'the value of e raised to 2.0 power is substituted for X#.
```

[Explanation]

EXP returns the value specifying e (the base of natural logarithms) raised to a power.
The constant e is approximately 2.718282.
The **EXP** function complements the action of the **LOG** function and it is sometimes referred to as the antilogarithms.

Refer to **LOG**.

FINISH (command)

[Function]

Allows the Z Axis goes down in the robot sequence.

 [Format]

FINISH_#<File number>

[Example]

FINISH #1

[Explanation]

When using SEQ~SEQEND command, the Z Axis down motion after the horizontal movement is allowed by FINISH command.

<File number> is specified by one of the number from 0 through 15. <File number> corresponded to the communication port (COM0, 1, 2, 3 or 8) must be designated by OPEN command in advance.

Refer to **SEQ~SEQEND** command.

FIX (function)

[Function]

Gets the integer value.

[Format]

FIX(<Numeric expression>)

[Example]

F=FIX(B/3)

[Explanation]

The maximum numeric value which does not exceed the value of <Numeric expression> is returned. FIX returns the value of the digits to the left of the decimal point.

FM (motion parameter memory)

[Function]

This memory content the value of F data of a position memory.

[Format]

< Variable >= REF (#<file number,FMm)   < file number > : 0,1,2,3,8

REF (# <file number>,FMm)=<variable

[Example]

```
MD1 = REF (#1, FM1) .......Applicable to retrieve
REF (#1, FM1) = MD1 .......Not applicable to substitute
```

[Explanation]

In order to get the stored F data of a position memory PM you can read it with the REF command. It is not possible to write back the F code to the PM directly by this memory.

Refer to **REF, PM**

FOR...TO...STEP~NEXT (command)

[Function]

Repeats a series of instructions from FOR through NEXT sentence as many as the given number of times.

[Format]

```
 FOR_<Variable name>=<Initial value>_TO_ <Final value>_ [ STEP increment>]
.....
NEXT_[<Variable name>]
```

[Example]

```
FOR J=0 TO 100 STEP 2
.....
NEXT J
```

[Explanation]

Repeats the program lines in FOR~NEXT loop according to the instruction in FOR sentence. The variable specified by <Variable name> must be an integer type or single precision real number type. The variable specified by <Variable name> is used as a counter and set to <Initial value> at first. The program lines after FOR sentence is executed until NEXT is encountered. Then the counter value is incremented by the amount specified by STEP. The counter value is compared with <Final value>. If the value is not greater than the final value, the program execution is back to the line after FOR sentence and the process is repeated. If the value of STEP is not specified, <Increment is regarded as + 1. If the condition is as follows, FOR~NEXT statement is not executed, but the program execution branches after NEXT sentence. The value of <Increment> can be minus.

(1) <Increment> is plus value and <Initial value> is greater than <Final value>.

(2) <Increment> is minus value and <Initial value> is less than <Final value>.

<Initial value> is substituted for variable. One FOR~NEXT loop may be placed in another FOR~NEXT loop. (Called " Nesting")

Note: 1. FOR and NEXT sentence must be corresponded each other. Therefore, NEXT sentence may be not set in IF sentence.

2. When nesting the FOR~NEXT loop;

1) <Variable name> must be different at each FOR~NEXT sentence.

2) One FOR ~ NEXT sentence must be completely inside another FOR~NEXT sentence.

3) The maximum nesting is 8 loops.

GETPRIORITY (command)

[Function]

Get the current priority of a job.

[Format]

```
Priority = GETPRIORITY("Job-name")
```

[Example]

```
PRIORITY% = GETPRIORITY("TESTJOB")
```

[Explanation]

GETPRIORITY function can get the current priority of the specified job. You can see the meaning of the priority at the explanation of <SETPRIORITY>.

[Errors]

If the specified job can not be found, the run time error 1A (Incorrect usage of command or function) occurs.

Refer to **SETPRIORITY**

GETROBNO (Function)

[Function]

Gets a robot number for the robot communication of a current job.

[Format]

<Variable>=GETROBNO( )

[Example]

```
ROBNO%=GETROBNO( )
```

[Explanation]

This function gets a robot number for the robot communication of a current job.

If the robot number has been set already, this function returns the value from zero to 999.
Just after STC power is reset or programs are downloaded, or after **CLEARROBNO** function is executed, this function returns –1 as the state without a robot number.

[Errors]

(Compiling error)
- No error occurs.

(Execution error)
- No error occurs.

Refer to **SETROBNO**, **SETROBNO**.

Note) **GETROBNO** is available for the following version.

- STC
  Available ROM version is 5.41 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.01 or later.
  In case of older version, an error occurs when compiling.

GLOBAL (command)

[Function]

Define the global variable.

[Format]

```
GLOBAL_<Variable name>[,<Variable name> [,...]]

                        or                      or

            <Array variable> <Array variable>

 <Array variable>=<Variable name>(<Max value of subscript>
[,<Max value of subscript>[,<Max value of subscript>]])
```

[Example]

GLOBAL ABC, MOJI $, XYZ % (20,5)

[Explanation]

The variables are normally independent at each job as local variables and the same variable names in a different job are not affected each other. This command is to define the variable common to all jobs as global variable. <Variable name> must be alphanumerics started with alphabet within 16 characters. The period (.) can be used in the name. The variable type can be designated using one of "Type declaration character" at the end of name. Refer to the section 3.7 " Variable".

Type declaration characters

$	... String type
%	... Integer type
!	... Single precision real number type
#	... Double precision real number type

If the type declaration character is not specified, the variable is regarded as a single precision real number type. The array of the global variable is defined by this command instead of DIM command. The format is similar to DIM and the array can be defined by three dimension at maximum. This command is valid in a job. In order to handle the same variable at different jobs, define the same <Variable name> as global variable in each job. If the same variable names which have been not declared are used at different jobs, these variables are regarded as local variables.

GOSUB (command)

[Function]

Branches to a subroutine.

[Format]

GOSUB_<Label>

[Example]

GOSUB * SUB 1

[Explanation]

The subroutine is an independent program from other program in one job and it is branched back by RETURN sentence. GOSUB command calls the subroutine program started with the line specified by <Label> in the same job program. When processing a subroutine, the GOSUB line is memorised in stack and the original line of the GOSUB command is called back at the completion of the process. A subroutine can be called from within another subroutine in the same job program. This is called the nesting of subroutine. There are 8 stacks for the nesting of subroutine. So, 8 nestings of subroutines are available at maximum.

If the total amount of stack is more than 8," Out of memory" error occurs.

Refer to **RETURN**.

GOTO (command)

[Function]

Branches the program to the specified line in the same job program unconditionally.

[Format]

GOTO <Label>

[Example]

GOTO *EXIT

[Explanation]

Branches the program execution to the line of <Label>. The program does not branch to other job programs.

Refer to **GOSUB, RETURN**.

HERE (reserved variable)

[Function]

HERE is a position memory which always keeps the current and actual position data of the robot.

[Format]

HERE

[Example]

MOVE #1,PTP,HERE+,,50

[Explanation]

The current position can be read out by REF command or can be used in combination with MOVE command. The memory HERE only stores the coordinates and the arm position.

		HERE		
X component	Y component	Z component	W component	ARM component

Refer to **MOVE, REF**

HEX$ (function)

[Function]

Returns a string converting decimal to hexadecimal value.

[Format]

HEX$(<Numeric expression>)

[Example]

A$=HEX$(X)

If the value of the variable X is 12 (&HC), the string "C" is substituted for A$.

[Explanation]

<Numeric expression> is converted to hexadecimal value and its string is returned. The value of <Numeric expression> must be integer value in the range -32768 to 32767 (or 0 to 65535).

The values of <Numeric expression> corresponded to the character string are;

The value of <Numeric expression>	Character string described by hexadecimal value
0 to 32767	0000 to 7FFF
- 32768 to - 1	8000 to FFFF
32768 to 65535	8000 to FFFF

Table 4.1

Refer to **STR$, VAL**.

HOLD ON/OFF (command)

[Function]

Specifies whether or not the robot holds (servolocks) the position after the completion of positioning.

[Format]

(1) HOLD_ ON_ #<File number>

(2) HOLD_OFF_#<File number>

(3) HOLD_ON_#<File number>, <Axis No>[,<Axis No>]

[Example]

HOLD ON #1

HOLD OFF #1

HOLD ON #1, 1,2,4

[Explanation]

The robot of the communication port specified by <File number> performs the servolock ON or OFF after the completion of positioning. <File number> is specified by one of the value from 0 through 15. <File number> corresponded to the communication port (COM 0, 1,2, 3 or 8) must be designated by OPEN command in advance. <Axis No> can be designated by X(A) Axis = 1, Y(B) Axis = 2, Z Axis = 3 and W Axis = 4 in the sequence when specifying specific axes to set ON or OFF the servolock.

Format (1) Servolocks all axes.

Format (2) Cancels the servolock.

Format (3) servolocks X, Y and W Axes.

Refer to **OPEN**

IF...THEN...ELSE...(ENDIF) (command)

[Function]

Judges the condition of logical expression.

[Format]

(1) IF_<Logical expression>_THEN_<Sentence1> [_ELSE_<Sentence2>]

(2) IF_<Logical expression>_THEN
..
<Sentence1>
...
[ELSE]
...
<Sentence2>
...
ENDIF

[Example]

(1) IF A$ = "Y" THEN GOTO *YES ELSE *NO

If the value of the variable A$ is "Y", then the program jumps to the line starting *YES. If the value of A$ is not "Y", the program jumps to the line *NO.

```
(2) IF TIM5 THEN
        A=B+C
    ELSE
      GOTO *EXIT
    ENDIF
```

If TIM5 has time-up, execute A=B+C. If TIM5 has not time-up, jump to *EXIT.

[Explanation]

This command makes a decision regarding program flow based on the result of logical expression. If <Logical expression> is true (-1), <Sentence1> clause is executed. If <Logical expression> is false (not -1), <Sentence2> clause is executed. The condition of time up for the timer is true. In the format (1) above, the multi-statement is not available for <Sentence1> or <Sentence2>.

In the format (2), the multi-statement can be used. However, any sentence can not be described at the right of the THEN, ELSE or ENDIF line. After <Sentence1> has executed, the program execution jumps to the next of ENDIF line.

Other IF sentence can be described in <Sentence1 > or <Sentence2> within one line.

INB, IND (operand)

[Function]

INB and IND design the inputs of the station controller STC which are represented by the field bus (Hirata Remote I/O or InterBus)

[Format]

INBm	m = 0 to 255	The value can be 0 or 1
INDm	m = 0 to 31	The value can be 0 to 255 or &H00 to &HFF

[Example]

WAIT INB5=1

A%=IND2+&H02

[Explanation]

INB: There are 256 input bit memories (INB) INB0 through INB255 and each can store the value 0 or 1 in order to express on/off status of input signal from external devices.

IND: There are 32 input data memories (IND) IND0 through IND31 in order to process consecutive 8 INBs by byte units.

Note:
Some signals from external devices may be intervened by electric noise or chattering. (Pulse signal which is occurred by bound between contacts.) After removing the chattering time by software, input signals are processed as INB or IND. Therefore, there is 20 msec delay (max) until a signal is stored into a memory.

IND0 consists of 8 bits from INB0 through INB7 and the relation is shown below.

IND0	INB7	INB6	INB5	INB4	INB3	INB2	INB1	INB0

The same relation is applied between INB0 to 255 and IND0 to 31 as shown in Table below.

Note:

Practically the number of input depends on the hardware structure such as the number of remote I/O units connected to a HAC robot controller. The address of each = memory can be specified indirectly by numeric expression or another memory in parentheses.

The Table shows the relation between INDs and INBs.

IND	**INB**							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	134	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
	128	64	32	16	8	4	2	1
	Each weight of INB corresponding to the value of IND							

Refer to **IF, WAIT**

INCLUDE (command)

[Function]

This command defines that there is a header file used in the job.

[Format]

```
INCLUDE "headerfile.bas"
```

[Example]

```
INCLUDE "header.bas"
```

[Explanation]

The command INCLUDE indicates that a header file is used in a program. The INCLUDE command has to be at the top of a job. It has to follow directly behind JOB NAME.

Refer to **DEFINE**

INPUT # (command)

[Function]

Retrieves data from a file and substitutes it for variable.

[Format]

INPUT_#<File number>,<Variable name>[,<Variable name>[,...]]

[Example]

INPUT #1, A, B$

Retrieves two data from the file 1 and substitutes them for the variables A and B$.

[Explanation]

<File number> must be the same number as specified by OPEN command. Type of <Variable name> must be corresponded to the data type retrieved from the file. If the data retrieved from the file is the value which is substituted for numeric variable, the data must be divided by space, comma (,) or carriage return (ASCII code 13). If the data is the value which is substituted for string variable, the data must be divided by comma (,) or carriage return. The line feed (ASCII code 10) after carriage return is ignored.

Refer to **OPEN, CLOSE, PRINT #.**

INPUT$ (function)

[Function]

Reads a specified length of character string from a specified file.

[Format]

INPUT$(<Number of characters>, #<File number>)

[Example]

A$=INPUT$(6, #2)

Retrieves 6 characters from the file 2 and substitutes them to A$.

[Explanation]

<File number> must be the same number as specified by OPEN command. The length of character string to be read is specified by <Number of characters> by byte unit. INPUT $ waits for the input of the specified number of characters into the file. If data has already been in the file, the specified characters are read. INPUT $ reads the control codes, which are not usually used as characters, and dividing symbols of file (space, comma and carriage return etc) as the character data.

Refer to **INPUT #, LINE INPUT #.**

INSTR (function)

[Function]

Searches for the specified character string and gets the position of it.

[Format]

INSTR(<Position>,<Character string 1>,<Character string 2>)

[Example]

```
J=INSTR(A$,"J")
```

Searches the character "J" in the string variable A$ and substitutes its position for the variable J.

[Explanation]

<Character string 2> is searched in <Character string I >. If <Character string 2> is found, its position is returned as the value. If <Character string 2> is not found, 0 is returned as the value. <Position> is an integer value of the character numbers of which position is to be searched first in <Character string 1>. If " " (Null string) is specified in <Character string 2>, the value of INSTR is the same as <Position>.

INT (function)

[Function]

Gets the integer value.

[Format]

INT(<Numerical expression>)

[Example]

```
A% = INT(B!/3)
B% = INT(-10.34)      '-11 substituted for B%
C% = INT(0.0)         '0 substituted for C%
D% = INT(3.68)        '3 substituted for D%
```

[Explanation]

INT returns the value of the digits to the left of the decimal point.
If <Numeric expression> has a negative value, **INT** returns the first negative integer less than or equal to <Numeric expression>, whereas **FIX** returns the first negative integer greater than or equal to <Numeric expression>.

```
B% = FIX(-10.34)
```

In this case, -10 is substituted for B%.
If <Numeric expression> has zero or positive value, **INT** and **FIX** returns the same value.

Refer to **FIX**.

IRB, IRD (operand)

[Function]

These operand represent the digital input of the robot controller. There are separate from the field bus inputs.

[Format]

IRBm	m : 0 -31	The value can be 0 or 1
IRDm	m : 0 - 4	The value can 0 to 255 or &H00 to &HFF

[Example]

A%=REF (#1,IRB2)

MD2=REF (#1,IRD0)

[Explanation]

IRB: There are 32 bit input memories from IRB0 through 31. Each can store the value 0 (off) or 1 (on) in order to express on/off condition. The actual available input bits depend on the hardware structure.

IRD: There are 4 input data memories from IRD0 through 3. Each can handle consecutive 8 IRBs as 1 byte parallel input data.

The relation of IRD and IRB are shown below.

IRD	IRB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24

Note: The processing speed to get the condition is slower than at INB and IND because the communication runs through serial communication.

Refer to **OPEN, REF**

JOB NAME (command)

[Function]

Defines a start line of a job and sets the job name.

[Format]

JOBANAME_"<Job name>"

[Example]

JOB NAME "ROBOT"

[Explanation]

<Job name> is expressed by the character string within 16 characters starting with an alphabet or numeric value. The character string must be enclosed by double quotation marks ("). Period (.), hyphen (-) and under bar (_) can be used in character string, but other symbols, Other characters and space can not be used. <Job name> must be always specified. The same <Job name> can not be used in another job which is linked later.

When designating <Job name> at JOB START / ON / OFF command, the job with the <Job name> defined by this command must be linked.

Refer to **JOB START/ON/OFF**.

JOB START/ON/OFF (command)

[Function]

Controls the job processing.

[Format]

(1) JOB_" <Job name>" _START

(2) JOB_"<Job name>"_ON

(3) JOB_"<Job name>"_OFF

[Example]

JOB "ROBOT" ON

[Explanation]

Starts, continues or interrupts the job processing specified by <Job name>.

Format
(1) Starts the processing of the specified job from first step. This is valid only for the job which is in JOB OFF state.
When this command is executed,

1) The nesting of **FOR~NEXT** and sub-routine is cleared.

2) The file opened in the job is closed.

3) Any error in the job is cleared.

Format (2) Continues the processing of the specified job. If the specified job is in **JOB OFF** state, the processing is resumed. However, if the specified job is in **JOB ON** or **START** state, this command is invalid.

Format (3) Interrupts the processing of the specified job. Since the program execution or the processing step is memorised at the occurrence of the interruption, the interrupted process is resumed by JOB ON command.

If the robot is moving, the motion is interrupted by this command. The motion is resumed by JOB ON command.

Note:

1. The same <Job name> specified by this command must be named in linked jobs.

2. A <Job name> is specified by JOB NAME command at the beginning of each job program.

3. JOB OFF command to its own job is allowed, but JOB ON or JOB START is required in another job.

4. All processes of the job including the interrupt control are terminated while JOB OFF command is working. Even if the condition of the interruption is satisfied during JOB OFF command, the status is ignored.

5. If JOB START command is executed to a job which is not in JOB OFF status, the error (error code 2Dh) occurs.

Refer to **JOB NAME**.

LEFT $ (function)

[Function]

Gets the characters of the specified length from its left side .

[Format]

LEFT$(<Character string>, <Numeric expression>)

[Example]

B$=LEFT$(A$,4)

The four characters from the left of the character string A$ are substituted for B$.

[Explanation]

The number of characters specified by <Numeric expression> from the left side of <Character string> is returned.

<Numeric expression> must be from 0 through 255. If the value of <Numeric expression> is greater than the number of <Character string>, all of <Character string> is returned. If the value of <Numeric expression> is 0, the null-string is returned

Refer to **RIGHT$, MID$.**

LEN (function)

[Function]

Gets the total byte number of character string.

[Format]

LEN(<Character string>)

[Example]

L=LEN("TEST")   4 is substituted for the variable L.

[Explanation]

The total byte number of character string is returned. The characters which are not output (character codes from O through 31, called control codes) and the space are also counted as the number of characters.

LET (command)

[Function]

Substitutes the specified values for a variable or memory.

[Format]

(1) [LET_]<Variable name>=<Expression>

(2) [LET_]TIM<Number>=<Numeric expression>

(3) [LET_] <Position memory>=<Data>

[Example]

(1) B$=XYZ"

(2) TIM31=2.5

(3) P12=(20, 25, 10,, LEFTY)

[Explanation]

LET is not a mandatory requirement.

Format (1)   Substitutes the right side for the left side.

Expression can be any type of numerics or character string. If the type is different between <Variable name> and <Expression>, the left side precision is applied to the substituted value.

Format (2)   Starts the timer.

<Number> of timer is specified by the value from 0 through 31. <Numeric expression> is setting time for timer specified by the value from 0.00 through 327.67. The minus value can not be specified. If JOB OFF command is executed while the timer is counting, the time count does not stop.

Format (3) Substitutes the values of the right side component for each component of the coordinate memory of the left side corresponded to the right side. <Data> is specified by one of the four below. Only position memory (P and PM) can be specified in the left side.

Data		The left side	The right side	Note
Position memory	P	O	O	It must be declared by DIMPOS command.
	PM	O	O	Uses REF command. One REF command can be used in one e.g.) MDD = REF ( ) AND REF (...) is not allowed.
HERE		X	O	
Describing coordinate component		X	O	Enclose entire description by parentheses and separate the value of component by comma (,).

Data can be expressed by addition or subtraction.

```
Format)          <Data> [+<Data>][+...]
                         or       or
                         -        -
```

Note: When describing the coordinate component in the right side, a part of component can be omitted (See format 3). In this case, the value of left side corresponded to the omitted part does not change.

LINE INPUT # (command)

[Function]

Retrieves a line data (within 255 bytes) to the character type variable.

[Format]

```
LINE_INPUT_#<File number>,<Name of character type variable>
```

[Example]

```
LINE INPUT #1,A$
```

[Explanation]

<File number> must be the same as the number specified by OPEN command. <Name of character type variable> is the name of character type variable which is substituted the data for.

The LINE INPUT # is to retrieve a line data before carriage return (ASCII code 13) in a file. The line feed (ASCII code 10) after carriage return is ignored.

Refer to **INPUT #, OPEN**.

<u>LOG (function)</u>

[Function]

Gets the natural logarithm.

[Format]

LOG(<Numeric expression>)

[Example] A=LOG(35/9)

The natural logarithm (logarithm to the base "e") is returned as the value. If <Numeric expression> includes double precision real number, the value is returned by double precision. In other case, the value is returned by single precision.

MB,MD (memories)

[Function]

MB and MD are battery backup memories.

[Format]

MBm=< value > m : 0 - 255 < value > = 0 or 1
MDm=< value > m : 0 - 255 < value > = 0 - 255 or &H00 - &HFF

[Example]

MB1=1
MB(MD2)=0

MD4=35
MD(MD7)=IND3

[Explanation]

MB: There are 256 bit memories (MB) 0 through 255, and each can store the value of 0 or 1.

MD: There are 256 data memories (MD) 0 through 255, and each can store a positive integer value 0 through 255. Data memories 0 through 31 correspond to bit memories (MB) 0 through 255, so that bit information can be handled in byte unit.

For example, MD0 consists of 8 bit MB0 through MB7. The relation is shown below.

MD0	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

These MBs are assigned the value by 8 bit binary as the contents (0 through 255) of MD.
When the value of MD0 is 0, 150 or 255, the value of MB is shown below.

	MSB							MLB
(The Value of MD0)	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
0	0	0	0	0	0	0	0	0
150	1	0	0	1	0	1	1	0
255	1	1	1	1	1	1	1	1
	128	64	32	16	8	4	2	1

Each weight of MB corresponding to the value of MD

Example) When the value of MD0 is 150.

150=128x1+64x0+32x0+16x1+8x0+4x1+2x1+1x0

The explanation above shows the relation between MD0 and MB0 to MB7. The same relation is applied between MB0 to 255 and MD0 to 31 as shown in table 2.3.

<u>The MD253 to 255 (data memory) are assigned to monitor the state of field-bus network. Do</u>

not use these MD as your memory.
See "2.2.9 How to Refer to Field-bus State from HARL-III Program" about MD253 to 255.

<Correspondence between MD and MB>

MD	MB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	134	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
32 255	128	64	32	16	8	4	2	1
	Each weight of MB corresponding to the value of MD							

Note:

1. The number of each memory can be specified indirectly by numeric expression or the value of another memory called as "pointer". The memory pointer must be enclosed in the parentheses.

2. Even if the power of the controller is reset, the contents of MB and MD are not initialised.

MID$ (function)

[Function]

Gets the specified part of character string in a character string or replace a part of character string.

[Format]

(1) MID$(<Character string>, <Numeric expression 1>[,<Numeric expression 2>])

(2) MID$(<Character string l>,<Numeric expression l>[,<Numeric expression 2>])=<Character string 2>

[Example]

(1) B$=MID$(A$,4,3)

Three characters from the 4th character at the left of the string A$ are substituted for B$.

(2) MID$(A$,4,3)="ABC"

ABC are substituted for three characters from the 4th character at the left of the string A$.

[Explanation]

MID$ command has two functions.

Format 1: The function in order to get character string.

Format 2: Substitution command to replace character string.

Format (1) The numbers of character string specified in <Numeric expression 2> are returned from the <Numeric expression 1>th character at the left of <Character string>. <Numeric expression 1> must be the value from 1 through 255 and <Numeric expression 2> must be from O through 255. If <Numeric expression 2> is omitted or the total numbers of <Character string> beginning with <Numeric expression 1> to the right is less than <Numeric expression 2>, all the character string beginning with the right of <Numeric expression 1>th character are returned. If the numbers of character in <Character string> is less than <Numeric expression 1>, a null-string is returned.

Format (2) The numbers of character string specified in <Numeric expression 2> beginning with <Numeric expression 1>th character of the left of <Character string 1> are replaced with the numbers of character specified in <Numeric expression 2> from the left of <Character string 2>.
If <Numeric expression 2> is omitted or the value of <Numeric expression 2> is more than the character numbers of <Character string 2>, all the characters of <Character string 2> are returned.
If the value of <Numeric expression 1> + <Numeric expression 2> - 1 is more than the character numbers of <Character string 1>, the value of (Character numbers of <Character string 1>) - <Numeric expression 1> + 1 is regarded as the value of

<Numeric expression 2> and the rest part of <Character string 2> is ignored. The value of <Numeric expression 1> must not be more than the character numbers of <Character string 1> and not be-O or less. Therefore, a null-string can not be specified in <Character string 1>.

Refer to **RIGHT$, LEFT$.**

MM (motion parameter memory)

[Function]

This memory contents the value of M data of a position memory PM.

[Format]

< variable > = REF (#< file number >,MMm)	m : 0 - 999
REF (#< file number >,MMm)=< variable >	< file number > : 0,1,2,3 or 8
	< Variable > : 00 to 99 or 255

[Example]

MD1 = REF (#1, MM1) ......MM data of PM1 is substituted for MD1.
REF (#1, MM1) = MD1 ......The value in MD1 is substituted for MM of PM1.

[Explanation]

In order to get the stored M data of a position memory PM you can read it with the REF command. It is not possible to write back the M code to the PM directly by this memory.

MM: MM is a M data stored in the robot controller. This memory can store the value from 0 through 99. However, the value of MM at end point (??) is assigned as 255.

The MM data can be substituted by REF command.

Refer to **REF, PM**

MOD (function)

[Function]

This function calculates the floating divide remainder of an integer division.

[Format]

< numerical expression1 > MOD < numerical expression2>

[Example]

A%=13.3 MOD 4	(A=13/4 = 3 Remainder 1)
B%=25,68 MOD 6,99	(B=26/7 = 3 Remainder 5)

[Explanation]

The function calculates the floating remainder of an integer division between <numerical expression 1 > and < numerical expression 2 >.

MOVE (command)

[Function]

Moves the robot to the specified position.

[Format]

```
MOVE_#<File number>,[<Motion pattern>],<Position data>[±X-Offset,Y-Offset, Z-Offset, W-Offset]
```

[Example]

```
MOVE #1,PTP, PM3
MOVE #2,PTP, PM(MW6)
MOVE #8,PTP, PM4+(0.1,0.2,-50,0)
MOVE #1,PTP, HERE

MOVE #1,PTP, P20
MOVE #1,PTP, P(MD2)+(,,+20)
```

[Explanation]

Moves the robot connected to the communication port specified by <File number> to the specified position. <File number> must be one of the value from 0 through 15. <File number> corresponded to the communication port (COM 0,1,2,3 or 8) must be specified by OPEN command in advance.

(1) Motion pattern

<Motion pattern> is specified by PTP or can be omitted. The robot moves by the pattern specified by M data at each position.

(2) <Position data>

<Position data> is to designate the position where the robot stops or passes.
Specify <Position data> by one of the followings. <Position data> can not be omitted.

<Position data>	Format and Explanation
Position data stored in robot controller	PMm (m = 0 to 999) PM is the position memories stored in the robot controller whose file number has been assigned as <#n>. (Example) PM123
Position memory	Pm The number of m is defined by DIMPOS command.
Components of coordinates	(<X component>, <Y component>, <Z component>, <W component>, <ARM component>) Specify the components required for each robot type. If a part of components is omitted, the current position is applied for the omitted component. Example) (100, 200, 50,, RIGHTY)
Current position of robot	HERE The current position of the robot is used.

<Position data> can be described by expression of addition (+).

Example)

PM0+(,, 10) ....Position where +10 mm of Z Axis from PM0

HERE+(,, -20) .... Position where -20 mm of Z Axis from the current position

Refer to **OPEN, DISABLE, PM, P, CALIB**

MRB,MRD (memories)

[Function]

These memories are located in the robot controller and with battery backup.

[Format]

```
REF (#< file number >,MRBm)=< value >    m : 0 -31       < value > 0 or 1
REF (#< file number >,MRDm)=< value >    m : 0 - 7       < value > 0 - 255 or &H00 - &HFF
                                                         < file number > : 0,1,2,3,8
< Variable > = REF (#< file number >,MRBm)
< Variable > = REF (#< file number >,MRDm)
```

[Example]

```
MB1 = REF (#1,MRB4)
B% = REF (#2,MRB23)
REF(#1,MRB1)=1

MD3=REF (#1,MRD0)
REF (#8,MRD2)=&H55
```

[Explanation]

MRB: There are 64 bit memories from MRB0 through 63, stored in a robot controller. Each can store the value 0 or 1.

MRD: There are 8 data memories, from MRD0 through 7, stored in a robot controller, and each memory can store the value from 0 through 255. Each of them is a 1 byte (8 bit) memory. Consecutive 8 MRBs are handled as one MRD.

The relation of MRD and MRB are shown below.

MRD	MRB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56

Refer to **REF**

MW (memories)

[Function]

These memories are 2 Byte memories with battery back up.

[Format]

MWm=< value > m : 0 - 255 < value > - 32768 to + 32767 or its hexadecimal equivalent

[Example]

```
MW200=5600
MW(MW2)=&HFFFF
```

[Explanation]

MW: There are 256 word memories (MW), and each can store an integer value - 32768 through + 32767. MWs occupy an independent memory area. There is no overlapping like between MDs and MBs.

Even if the power of the controller is reset, the contents of MW are not initialised.

The MW254 to 255 (data memory) are assigned to monitor the state of field-bus control. Do not use these MW as your memory.
See "2.2.9 How to Refer to Field-bus State from HARL-III Program" about MW254 to 255.

NETCLOSE (Function)

[Function]

Closes the network communication.

[Format]

NETCLOSE(<Variable>)

[Example]

```
STATION%=2                  'station no. #2
NID%=NETOPEN(STATION%)  'open network communication for station #2
        •
        •
NETCLOSE(NID%)              'terminate network communication
```

[Explanation]

NETCLOSE function closes the network communication that has the network identifier specified as <Variable>. The specified network identifier must have the number that has assigned by **NETOPEN** function.

[Errors]

(Compiling error)

- **NETCLOSE** does not have a returned value. If **NETCLOSE** is substituted for a variable, the error “Type mismatch” occurs.
- If an expression such as a numerical constant instead of a variable is specified to <Variable> (network identifier) field, the error “Bad argument type of function” occurs.

(Execution error)

- If the specified network identifier is invalid, the error “Incorrect usage of command or function” occurs.

Refer to **NETOPEN**.

Note) **NETCLOSE** is available for the following version.

- STC
  Available ROM version is 5.40 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.00 or later.
  In case of older version, an error occurs when compiling.

NETOPEN (Function)

[Function]

Opens the network communication.

[Format]

<Variable>=NETOPEN(<Numeric expression>)

[Example]

```
STATION%=2                    'station no. #2
NID%=NETOPEN(STATION%)  'open network communication for station #2
```

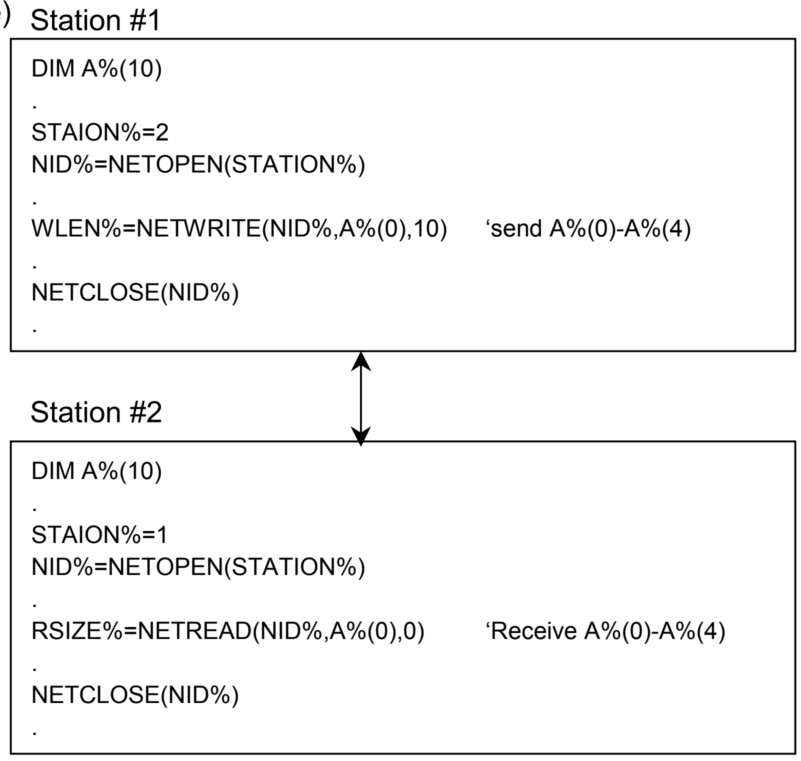
[Explanation]

When you use **NETOPEN** in HARL-III program running in STC, it is necessary that you must create the network definition and download it to STC. Refer to operation manual of HARL-III Compiler about details.

NETOPEN function opens the network communication for the station with the number specified to <Numeric expression>.
Station number must be 0 to 127.
NETOPEN returns with the value of the network identifier. **NETREAD**, **NETWRITE** needs this network identifier to read from or write to the network. NETCLOSE also needs this network identifier to close the network communication.

When communicating with the network, **NETOPEN** has been executed only one time for all jobs. If **NETOPEN** is executed twice without closing, an execution error occurs.

Typical usage)



Station #1

```
DIM A%(10)
.
STAION%=2
NID%=NETOPEN(STATION%)
.
WLEN%=NETWRITE(NID%,A%(0),10)     'send A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

Station #2

```
DIM A%(10)
.
STAION%=1
NID%=NETOPEN(STATION%)
.
RSIZE%=NETREAD(NID%,A%(0),0)       'Receive A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

The number of opened stations at the same time is restricted to 16.

[Errors]

(Compiling error)

- If the specified station number is the numerical constant with the value out of 0 – 127, the error “Illegal value of argument” occurs.

(Execution error)

- If the specified station number has the value out of 0 – 127, the error “Incorrect usage of command or function” occurs.
- If the specified station number is the own station number, the error “Own station number specified” occurs.
- If the network definition of the specified station is not found, the error “Network CR(Communication Reference) undefined” occurs.
- If **NETOPEN** has been already executed for the specified station, the error “Network already opened” occurs.
- If more than 16 stations is opened simultaneously, the error “Network open overflow” occurs.

Refer to **NETREAD**, **NETWRITE**, **NETCLOSE**.

Note) **NETOPEN** is available for the following version.

- STC
  Available ROM version is 5.40 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.00 or later.
  In case of older version, an error occurs when compiling.

NETREAD (Function)

[Function]

Reads data from the network communication opened by NETOPEN.

[Format]

<Variable#1>=NETREAD(<Variable#2>,<Variable#3>,<Numeric expression>)

[Example]

```
STATION%=2                        ‘station no. #2
NID%=NETOPEN(STATION%)  ‘open network communication for station #2
SIZE%=NETREAD(NID%,DATA%,0) ‘read data to DATA%
```

[Explanation]

When you use **NETREAD** in HARL-III program running in STC, it is necessary that you must create the network definition and download it to STC. Refer to operation manual of HARL-III Compiler about details.

The network identifier assigned by **NETOPEN** is specified to <Variable#2>.
A variable for received data is set is specified to <Variable#3>. You can specify a variable of character string.
The following option flag is specified to <Numeric expression>.
&H0000: Waiting for receiving any data without execution of next step.
&H0001: If received data not found, the next step is executed immediately.

Maximum size of received data is 50 bytes.

If the element of array like A%(0) is specified to <Variable#3>, the received data is set to the sequential area in which the first element is the specified element of array. You cannot specify the name of array.
Example)

```
    DIM X%(10)
    SIZE%=NETREAD(NID%,X%(1),0)    ‘received data is set to X%(1),X%(2)...
    SIZE%=NETREAD(NID%,X%,0)       ‘compiling error
```

Note) If the size of received data is bigger than the size of setting variable, area of other variable is destroyed.

NETREAD returns with the value of the size of received data.
Zero of data size means that data is not received. In case that the value with the bit #0 ON is specified to option flag <Numeric expression>, the size of received data indicates that data has been received or not.
Example)

```
    *LOOP
        SIZE%=NETREAD(NID%,DATA%,1)        ‘next step even if data is not
        received
        IF SIZE%=0 THEN GOTO *LOOP         ‘read again when not received
```

Typical usage) Station #1

```
DIM A%(10)
.
STAION%=2
NID%=NETOPEN(STATION%)
.
WLEN%=NETWRITE(NID%,A%(0),10)     'send A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

Station #2

```
DIM A%(10)
.
STAION%=1
NID%=NETOPEN(STATION%)
.
RSIZE%=NETREAD(NID%,A%(0),0)       'Receive A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

[Errors]

(Compiling error)

- If an expression such as a numerical constant instead of a variable is specified to <Variable#2> (network identifier) field, the error “Bad argument type of function” occurs.
- If an expression such as a numerical constant instead of a variable is specified to <Variable#3> (received area) field, the error “Illegal function call” occurs.

(Execution error)

- If the specified network identifier is invalid, the error “Incorrect usage of command or function” occurs.
- If the specified network identifier is not opened, the error “Network not opened” occurs.
- If the network definition of the specified network identifier is not found, the error “Network CR(Communication Reference) undefined” occurs.

Refer to **NETOPEN**, **NETWRITE**, **NETCLOSE**.

Note) **NETRAED** is available for the following version.

- STC
  Available ROM version is 5.40 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.00 or later.
  In case of older version, an error occurs when compiling.

NETWRITE (Function)

[Function]

Writes data to the network communication opened by **NETOPEN**.

[Format]

<Variable#1>=NETWRITE(<Variable#2>,<Variable#3>,<Numeric expression>)

[Example]

```
DATA%=1                      ‘set sending data
STATION%=2                   ‘station no. #2
NID%=NETOPEN(STATION%)  ‘open network communication for station #2
SIZE%=NETWRITE(NID%,DATA%,2) ‘write data
```

[Explanation]

When you use **NETWRITE** in HARL-III program running in STC, it is necessary that you must create the network definition and download it to STC. Refer to operation manual of HARL-III Compiler about details.

The network identifier assigned by **NETOPEN** is specified to <Variable#2>.
A variable of sending data is specified to <Variable#3>. You can specify a variable of character string.
The size of sending data is specified to <Numeric expression>.

Maximum size of sending data is 50 bytes.

If the element of array like A%(0) is specified to <Variable#3>, the data of the sequential area in which the first element is the specified element of array is sent. You cannot specify the name of array.
Example)

```
DIM X%(10)
SIZE%=NETWRITE(NID%,X%(1),6)  ‘sends data in X%(1),X%(2),X%(3)
SIZE%=NETWRITE(NID%,X%,6)     ‘compiling error
```

NETWRITE returns with the value of the size of sent data. Returned size is the same as the specified sending size normally.

Typical usage) Station #1

```
DIM A%(10)
.
STAION%=2
NID%=NETOPEN(STATION%)
.
WLEN%=NETWRITE(NID%,A%(0),10)     'send A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

Station #2

```
DIM A%(10)
.
STAION%=1
NID%=NETOPEN(STATION%)
.
RSIZE%=NETREAD(NID%,A%(0),0)       'Receive A%(0)-A%(4)
.
NETCLOSE(NID%)
.
```

[Errors]

(Compiling error)

- If an expression such as a numerical constant instead of a variable is specified to <Variable#2> (network identifier) field, the error “Bad argument type of function” occurs.
- If an expression such as a numerical constant instead of a variable is specified to <Variable#3> (sending area) field, the error “Illegal function call” occurs.
- If a numerical constant out of 0 to 234 is specified to <Numerical expression> (data size) field, the error “Illegal value of argument” occurs.

(Execution error)

- If the specified network identifier is invalid, the error “Incorrect usage of command or function” occurs.
- If the specified network identifier is not opened, the error “Network not opened” occurs.
- If the network definition of the specified network identifier is not found, the error “Network CR(Communication Reference) undefined” occurs.
- If the specified data size is out of 0 to 50 bytes, the error “Network writing size error” occurs.

Refer to **NETOPEN**, **NETREAD**, **NETCLOSE**.

Note) **NETWRITE** is available for the following version.

- STC
  Available ROM version is 5.40 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.00 or later.
  In case of older version, an error occurs when compiling.

NOT (function)

[Function]

This function inverts the numerical expression. (Negation)

[Format]

NOT <numerical expression>

[Example]

1) IF NOT (A = 0) THEN GOTO *GO2
   If A is not 0, jump the program to the label *GO2.

2) - X = NOT X - 1
   A minus number is complement of 2 adding 1 to its complement of 1.

[Explanation]

X	NOT X
1	0
0	1

ON ERROR GOTO (command)

[Function]

Defines the start line to be branched when an error occurs.

[Format]

ON_ERROR_GOTO_<Label>

[Example]

ON ERROR GOTO *ONERR

[Explanation]

If there is no error processing program, the program execution will be terminated at the: occurrence of error. This command is to branch the program to the error processing routine started with the specified line without stopping program execution when an error occurs. The start line of error processing routine is specified by <Label> in the same job program. When an error occurs, its error code is set into the reserved variable, ERR. Therefore, the type of the error can be checked in the error processing routine by using ERR. The error processing sub-routine goes back to the main program by RESUME command.

Note: If an error occurs in the error processing routine, the program execution is stopped.

Refer to **ERR, ERROR, RESUME**, Error code list.

ON GOTO/ON GOSUB (command)

[Function]

Branches the program execution to the specified line.

[Format]

```
ON_<Expression>_GOSUB_<Label>[,<Label>...]
ON_<Expression>_GOTO_<Label>[,<Label>...]
```

[Example]

```
ON A GOSUB *SUB 1,*SUB2,*SUB3,*SUB4
ON A GOTO *SUB 1,*SUB2,*SUB3,*SUB4
```

[Explanation]

The value of <Expression> will be used for branching the program to the label in the list. The order of <Label> is corresponded to the number started with 1. For example, if the value of expression is 3, the third <Label> will be the destination of the branch. The program can not be branched to other jobs.

In ON...GOSUB sentence, each label in the list must be the first line of a sub-routine. If the value of expression is minus, "Illegal function call" error occurs. If the value is 0 or more than the number of items in the list, the program control continues with the next line and the error does not occur.

Refer to **GOSUB, GOTO**.

OPEN "COM

[Function]

Opens a communication file.

[Format]

OPEN_" COM[<Port number>]: [<Parameter>]" *A*S #<File number>

[Example]

OPEN "COM1:4800,E,7,1" AS #1

[Explanation]

In order to input/output the communication port specified in <Port number> as the file, a buffer is allocated and <File number> is specified to it. After this operation, the access to this file is executed by specifying <File number>.

(1) <Port number>

<Port number> is specified by the value, 0, 1,2, 3 or 8.

Port number	Type	Explanation
0	Dual port RAM communication	Faster communication speed with robot controller. Available only with the controller connected by dual port RAM (only HAC-2xx)
1	RS232C communication port	Versatile communication port.
2 3	Optional RS232C communication port	Versatile communication port. HPC-717 type board normally has these ports. For other type board, hardware expansion is required.
8	Host communication port*	Communication port for host computer. But able to communicate with a robot.
(9)	Programming console port	Can not be specified.

Table 4.4

* If **OPEN** command is executed to the port 8, the communication protocol for the host communication (HRCS-VI) is prohibited and the communication port 8 can be used as the versatile port.

If <Port number> is omitted, COM 1 is selected.

(2) <Parameter>

When the dual port RAM communication port is selected, <Parameter> is not necessary, because there is no <Parameter> in COM0. <Parameter> is to set the conditions for RS232C communication. Set the condition dividing by comma (,) in order below.

<Speed>, <Parity>, <Data>, <Stop>

Speed	An integer constant specifying the bit transmission speed 1200, 2400, 4800, 9600, 19200, 38400 (bps: bit per second) * 19200 and 38400 is available when you use STP ROM version 5.30 or later and use HARL-III Compiler for Windows version 1.11 or later. (HARL-III Compiler for DOS cannot compile this value.)
Parity	A character specifying the parity for transmission and receipt O (Odd): Odd transmit parity, odd receive parity checking. E (Even): Even transmit parity, even receive parity checking. N (None): No parity
Data	An integer constant indicating the number of data bit.7,8
Stop	An integer constant indicating the number of stop bit. 1: If the speed is 4800 and less, set 1 stop bit. 2: If the speed is 9600 and more, set 2 stop bit.

Table 4.5

The set <Parameter> is valid only for the communication port of <Port number> specified by
OPEN "COM command.

When a part of <Parameter> setting is omitted, specify as below.

1) When omitting <Data> and <STOP>;

```
9600,E
```

2) When setting <STOP> only;

```
,,,1
```

When a part or all setting are omitted, the values set before is valid.

Note: Even if <Parameter> is specified to COM0, the set value is ignored.

(3) <File number>

<File number> is specified by one of the value from 0 to 15. The specified <File number> must be the same as the <File number> of the communication commands (e.g.: **MOVE**) which are executed through the file.

(4) Note when opening several files at the same time, some files can be opened in the program at the same time by using **OPEN** commands.

In this case, pay attention to the followings.

1) The same <Port number> can not be opened at the same time.

2) The same <File number> can not be opened at the same time.

Refer to **CLOSE**.

OR (logical operator)

[Function]

This operator is the logical addition. Logical operation is used for checking several conditions, bit operation or pool operation. Logical operation gives 0 or 1 to each bit as result.

[Format]

< numerical expression > OR < numerical expression >

[Example]

1) IF X < 0 OR 99 < X THEN GOTO *GO1
   If X is larger than 99, jump the program to the label *GO1.

2) 12 OR 11 = 15 12 = (1100)2, 11 = (1011) 2
   Therefore, 12 OR 11 = (1111)2= 15

3) 32767 0R -32768 = - 1 32767=(0111111111111111)2
   -32768 = - (1000000000000()00) 2
   Therefore,32767 0R-32678 = (1111111111111111)2= -1

[Explanation]

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

Logical operation is also used for changing program flow as relational operator. In this case, logical operator can be connected by one relational operator and more.

Before operation, logic operator converts the numerics to integer expressed by complement from - 32768 through + 32767. If the result is beyond the range, overflow occurs. When only0 (falsehood) or -1 (true) is given, logic operator gives 0 or -1 as result. In specified logical operation, it is operated in bit unit. Therefore, logical operator can check the byte data by referring bit pattern. For example, OR operator can mix two bit patterns for making a binary number.

Refer to **AND, XOR**

<u>ORB,ORD (operand)</u>

[Function]

These operand represent digit output of the robot controller,

[Format]

REF (#< file number >,ORBm)=< value >	m : 0 - 31	< value > 0 or 1
REF (#< file number >,ORDm)=< value >	m : 0 - 4	<value > 0 - 255 or &H00 - &HFF
		< file number > : 0,1,2,3,8

[Example]

REF (#1,ORB2)=1

REF (#8,ORD0)=&HFF

[Explanation]

ORB: There are 32 output memories from ORB0 through 31 and each can store the value 0 or 1 in order to turn on (1) or off (0) the parallel output signal of each robot controller. The actual available output bits depend on the hardware structure.

ORD: There are 4 output data memories from ORD0 through 3. Each can process consecutive 8 ORBs as 1 byte parallel output data.

The relation of ORD and ORB are shown below.

ORD	ORB							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24

OUTB, OUTD (operand)

[Function]

These operands represent the digital outputs of the field bus at station controller.

[Format]

OUTBm=< value >	m : 0 - 255	< value > 0 or 1
OUTDm=< value >	m : 0 - 31	< value > 0 - 255 or &H00 - &HFF

[Example]

OUTB2=1
OUTB(MD3)=0

OUTD3=MD6
OUTD(MD7)=&HAA

[Explanation]

OUTB: There are 256 output bit memories (OUTB) OUTB0 through 255 and each can store the value 0 or 1 in order to turn on (1) or off (0) the output signal to an external device.

OUTD: There are 32 output data memories (OUTD) OUTD0 through 31 and each can process consecutive 8 OUTBs as Ⅰ Byte output data.

OUTD0 consists of 8 bits from OUTB0 through 7 and then relation is shown below.

OUTD0	OUTB7	OUTB6	OUTB5	0UTB4	OUTB3	OUTB2	OUTB1	OUTB0

The same relation is applied between OUTB0 to 255 and OUTD0 to 31 as shown in Table below.

Table shows the relation between OUTDs and OUTBs.

OUTD	**OUTB**							
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	134	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	255	254	253	252	251	250	249	248
	128	64	32	16	8	4	2	1
	Each weight of OUTB corresponding to the value of OUTD							

Note:

Practically the number of input/output depends on the hardware structure such as the number of remote I/O units connected to a HAC robot controller. The address of each = memory can be specified indirectly by numeric expression or another memory in parentheses.

The ON condition of the outputs are reset when power on.

P, PDF, PDM, PX, PY, PZ, PW (memory)

[Function]

P and its components is a memory of coordinates and motion parameters which are located in the station controller. In opposite to PM each component of P can be individually manipulated by calculations or settings.

[Format]

See below

[Example]

See below

[Explanation]

P: P is the memory to handle the position data of robot. P can not be used until the total amount of P is defined. The maximum numbers available by P is 608 (P0 through P607).

When position memories are required, define the position memories by DIMPOS command. E.g., 20 memories are required then use DIMPOS 20 command P0 through P19 can be used. If the amount of position memories defined by DIMPOS command is different among the jobs, the memories defined in each job are applicable in the job.

A P consists of 8 components as below.

Pm							
PXm	PYm	PZm	PWm	ARMm	PDMm	PDFm	Sm
X component	Y component	Z component	W component	ARM component	M data component	F data component	S data no direct access

"m" represents P number. In case of indirect designation using a variable or expression as a pointer instead of an integer, enclose such a pointer with parentheses.

PX, PY, PZ, PW	: Each of them is a memory for the component of X, Y, Z and W and these are long word (4 byte) memories which can store the value - 2147483.648 through + 2147483.647.
ARM	: ARM is the memory for the component of P and can store LEFTY or RIGHTY. ARM component is available for SCARA type robot mechanism.
PDM	: PDM is the memory for the M-code data which can store value 00 to 99 and ??=&HFF (= end point)
PDF	: PDF is the memory for the F-code data which can store value 00 to 99
S	: S code cannot be set directly. S code of P can only by set by REF command

Even if the power of the controller is reset, the contents of P and its components are not initialised.

[How to use P]

Each data of P components is created and used with REF command like the samples below.

<u>Example 1)</u>

Substitute a robot position memory (PM) for a P.

P10 = REF (#1, PM110)

After the program above has been executed, the following data are substituted for each component of P10.

PX10	X Axis data of PM110
PY10	Y Axis data of PM110
PZ10	Z Axis data of PM110
PW10	W Axis data of PM110
ARM10	L or R data of PM110
PDM10	M data of PM110
PDF10	F data of PM110

S code of PM110 are substituted for the internal area of P10.

PX10, PY10, PZ10, PW10; PDM and PDF can be handled as numerical variables in the program.

ARM10 can be specified by ARM10 = RIGHTY or ARM10 = LEFTY.

The M data, F code and S code of PM110 are substituted for the internal area of P10, but only S code data in P10 can not be read or write directly. When setting those data, refer to Example 4) below.

<u>Example 2)</u>

Substitute the current position data of the robot for a P.

P0 = REF (#1, HERE)	Substitute the current position for P0.
PZ0 = PZ0 - 10	Set Z Axis data of P0 at 10 mm upper than the current position.
PW0 = 200	Set 200 degree as W Axis data of P0.
MOVE #1, PTP, P0	Move the Z, W Axes to the modified position.

* When the current position of the robot is substituted, M = 1, F = 99 and S = 0 are automatically stored in the internal area of the P.

Example 3)

Substitute the robot current position data of the robot for a PM.

P0 = REF (#1, HERE) Substitute the current position for P0. REF (#1, PM100) = P0 Substitute the data in P0 for PM100.

* When the program above is executed, MM = 1, FM = 99 and SM - 0 are stored in PM100.
When setting a specific data to MM of PM100, program like the sample below.

REF (#1, MM100) = 50 Set 50 as M data of PM100.

Example 4)

Copy the robot position memory PM to another PM.

```
 P0 = REF (#1, PM100)     Substitute the data of PM100 for P0.
 P1 = REF (#1, PM200)     Substitute the data of PM200 for P1.
 PX1 = PX0                Copy X, Y, Z, W and ARM data of P0 to P1.
 PY1 = PY0
 PZ1 = PZ0
 PW1 = PW0
 ARM1 = ARM0
PDM1=PDM0
PDF1=PDF0
 REF (#1, PM200) = P1     Substitute P1 data for PM200.
```

By the program above, X, Y, Z, W. M code, F code and ARM data of PM100 are copied to PM200, but the S code stored in PM200 remains at PM200.

PAI (constant)

[Function]

Gets the value of π.

[Format]

PAI

[Example]

```
A# = PAI          ‘the value of π substituted for A#
```

[Explanation]

PAI is the arithmetic constant of π.
The value of π is approximately 3.1415927.

PM (memory)

[Function]

This memory is located in the robot controller (HNC) and stores the coordinates and motion parameter of a position.

[Format]

PMm		m : 0 - 999

[Example]

PM200
PM(MD)
PM(MW)

[Explanation]

PM: PM is a position memory stored in the robot controller. One PM consists of the components below.

PMm							
X component	Y component	Z component	W component	ARM component	MM component	FM component	SM component

Each component of PM can not be handled directly. When setting each component in the program, use REF command to substitute the data for P.

Note:

The value of "m" can be specified indirectly with a Memory Pointer.

Refer to **MOVE, REF, P**

PRINT # (command)

[Function]

Writes data to a file.

[Format]

```
PRINT_#<File number>,<Expression>[, <Expression>...][, ]
                                  or                  or
                                  ;                   ;
```

[Example]

PRINT #2,A;B;C

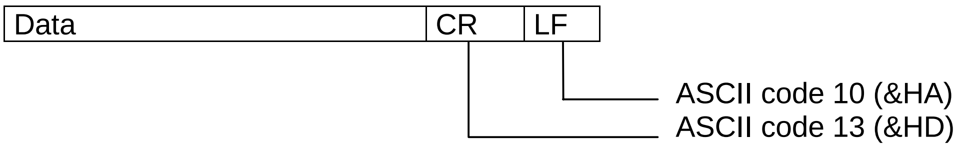
PRINT #1,A$;",";B$

[Explanation]

Writes the character strings or numerics specified in <Expression> to the file. <File number> must be the same as the number specified by OPEN command. The character or numeric expression to be written into the file is specified in <Expression>.

(1) File format

The data of the character strings and numeric values are written by ASCII code and the format below.



Example 1: Writes character string "ABC" to the file #1.

PRINT #1,"ABC"

The file format is as follows.

ABC	CR	LF

(CR/LF attached automatically)

Example 2: Writes numerics "123" to the file #1.

PRINT #1,123

The file format is as follows.

°123°	CR	LF

Space (For dividing)

Sign (In case of 0 and more, enter space (°).
In case of minus, enter " - " sign.)

* The numerics in a file are processed with a sign before the numerics and a space after them.

(2) Specification of <Expression>

One or some <Expression> can be specified. When specifying several <Expression>, connect <Expression> with semicolon (;) or comma (,).

Example 1: Connecting 5 <Expression>s with semicolons.

PRINT#1, 123; "ABC";",";"VWXYZ";-9876

The file format is as follows.

°123°ABC, VWXYZ -9876°	CR	LF

* Semicolon (;) is to write the value of <Expression> continuously

Example 2: Connecting 2 <Expression>s with comma.

PRINT #1,"ABC",-9876

The file format is as follows.

ABC°°°......°°° -9876A	CR	LF

' 14 characters '

* There is a block whose unit is 14 characters in a file data. Each value of <Expression> before comma is assigned the block. When the value of <Expression> is more than 14 characters, the required amount of blocks are applied for the value.

Example 3: Connecting 3 <Expression>s with semicolons and commas.

PRINT #1,123;"ABC",-9876

The file format is as follows.

°123° ABC°°°....°°°-9876°	CR	LF

' 14 characters '

Example 4: Terminates the end of command by a semicolon.

PRINT #1,123;"ABC";",";"VW";

The file format is as follows.

°123°ABC,VW

This file format does not have CR/LF. If there is a PRINT # command which is not terminated with a semicolon or a comma after this file format,

PRINT #1, "XYZ"; -9876

the file data are added as below.

°123°ABC,VWXYZ-9876°	CR	LF

Example 5: Terminates the end of command by a comma.

PRINT #1,"ABC",

The file format is as follows.

ABC°°°°°°°°°°°°°°°

' 14 characters '

The file format does not have CR/LF like the example 4 above. If there is a PRINT # command which is not terminated with a comma,

```
PRINT #1,-9876
```

the file data are added and the file format becomes as below.

ABC°°°°°°°°°°-9876A°	CR	LF

(3) Delimiter

The data range to be read from a file is divided by a delimiter. There are 3 type delimiter, carriage return (CR), comma (,) and space (°).

Delimiter	Function
Carriage return (CR)	Delimiter for one data line The line can be read by LINE INPUT # command
Comma (,)	Delimiter for one value (Numeric or character string) The value can be read by INPUT # command.
Space (°)	Delimiter for one value (Numerics) When the value read by INPUT # command is numerics, a space is worked as the delimiter. More than one space are available as a delimiter. A comma or a carriage return after the space is ignored.

Table 4 6

Note:

1. Line feed (LF) after carriage return (CR) is ignored.

2. Data must be within 255 characters. Space before or after numerics, minus symbol (-), comma (,), carriage return (CR), line feed (LF) and other control codes are counted as one character.

Refer to **OPEN, CLOSE**

PULSE (command)

[Function]

Sets the value of the variable to the specified value for the specified time.

[Format]

PULSE<Variable>=<Expression 1>,<Expression 2>

[Example]

(1) PULSE OUTB0=1, 2	Set OUTB0 to 1 for 2 seconds.
(2) PULSE OUTD1=&HFF, 3	Set OUTD1 to &HFF (255 in decimal) for 3 seconds.
(3) PULSE A%=10.5	Set A% to 10 for 5 second.

[Explanation]

The value of <Variable> is set to <Expression 1> for <Expression 2> seconds. After the time set by <Expression 2> has been passed, the value before the execution is reset.

This command is not applicable for the character type variable.

The time set for <Expression 2> is 0.00 through 327.67 (seconds).

Refer to **OUTB, OUTD**

REF (command)

[Function]

Handles the data stored in a robot controller.

[Format]

REF(#<File number>,<Data>)

[Example]

(1) MD3=REF(#1,IRD1)   One byte input data of the robot is substituted for MD3.

(2) REF(#1,PM5)=P20   The position data of P20 is substituted for PM5.

[Explanation]

The data stored in a robot controller is handled through communication. Specify the value of <Data> of the robot controller which is connected to the communication port specified by <File number>. <File number> must be one of the value from 0 to 15. <File number> corresponded to the communication port (COM 0,1,2,3 or 8) must be specified by OPEN command in advance. <Data> is to specify status, current position and internal memories of robot controller. Refer to section 2.2.5 to 2.2.8.

Only one REF command can be used at one sentence.

MB0=REF(#1, IRB0)AND REF(#1, IRB1) .... Not allowed.

Refer to **P, PM, IRB, IRD, ORB, ORD, MRB, MRD, HERE, FM, MM, SM**

REM (command)

[Function]

Defines the comment line.

[Format]

```
REM[<Comment>]
```

[Example]

```
REM*****MAIN*****
```

[Explanation]

REM sentence is not an executable statement and can be a comment line without giving the effect to program execution. When the program is listed, the entered contents are output as it is. Apostrophe (') can be used instead of REM. The program line after REM sentence does not accept the multi-statement. When expressing REM sentence after other sentences, one or more spaces before REM sentence are sufficient instead of dividing the sentences by colon (:).

RESUME (command)

[Function]

Completes the error processing routine and continues program execution.

[Format]

(1) RESUME

(2) RESUME_NEXT

(3) RESUME_<Label>

[Example]

RESUME *START1

[Explanation]

Continues program execution after an error recovery procedure is performed. After this command is executed, the error code of ERR is cleared to 0. (No error)

Format (1) Resume the program execution at the sentence which caused error.

Format (2) Resume the program execution at the next sentence which caused error.

Format (3) Resume the program execution at the line specified in <Label>.

If RESUME command is executed without error, the error (error code 12H) occurs.

Refer to **ON ERROR GOTO, ERR, ERROR code list**.

RETURN (command)

[Function]

Terminates the subroutine or interruption processing routine and continues the original program execution.

[Format]

```
 RETURN_[<Label>]
```

[Example]

```
RETURN
RETURN *START0
```

[Explanation]

RETURN sentence is used for termination of subroutine or interruption processing routine. When there is RETURN sentence in subroutine program, the execution of subroutine is terminated and the program following one where GOSUB command was executed is resumed.

If there is a RETURN sentence in a program for interruption processing routine, the processing is terminated and the program which waited due to the interruption is resumed. One and more RETURN sentences in one subroutine or interruption processing routine are available. When activating a subroutine or interruption processing routine, one stack is used to memorise the original position.

When RETURN is executed, the stack is reduced one. RETURN command can return to the line designated by <Label>. When using several subroutines or calling a subprogram of GOSUB command from FOR NEXT command, the destination of RETURN may not be the stack which was used at branching. If the destination of RETURN is not necessary to be the specified label, do not specify the label.

Note: If a RETURN command is used not in a subroutine or interruption processing routine, (used stack is 0), "RETURN without GOSUB" error occurs.

<u>RIGHT$ (function)</u>

[Function]

Returns the specified length characters from the right of the character string.

[Format]

RIGHT$(<Character string>,<Numeric expression>)

[Example]

B$=RIGHT$(A$,4)

[Explanation]

The number of characters specified in <Numeric expression> is returned from the right of <Character string>. <Numeric expression> must be from 0 to 255. When the value of <Numeric expression> is greater than the character number of <Character string>, all <Character string> are returned. When the value of <Numeric expression> is 0, null-string is returned.

Refer to **LEFT$, MID$.**

SELECT CASE (command)

[Function]

Executes one of several process blocks according to the expression.

[Format]

SELECT_CASE_<Expression>
CASE_<Judge expression 1>
<Block 1>
CASE_<Jugde expression 2>
<Block 2>
CASE_ELSE
<Block n+1>
END_SELECT

[Example]
(1)

```
SELECT CASE_A%*2
CASE 10,12,20 TO 100,IS <= 200
                                        If A%*2 is from 10, 12 or 20 to 100 or 200
 <Block 1>                              and less, Block 1 is executed. (Judge 1)

 CASE 250,N%
                                        If A%*2 is 250 or the same as N%, Block 2
 <Block 2>                              is executed. (Judge 2)

 CASE ELSE
                                  If Judge 1 and 2 do not match, Block 3 is
 <Block 3>                        executed.

END SELECT
```

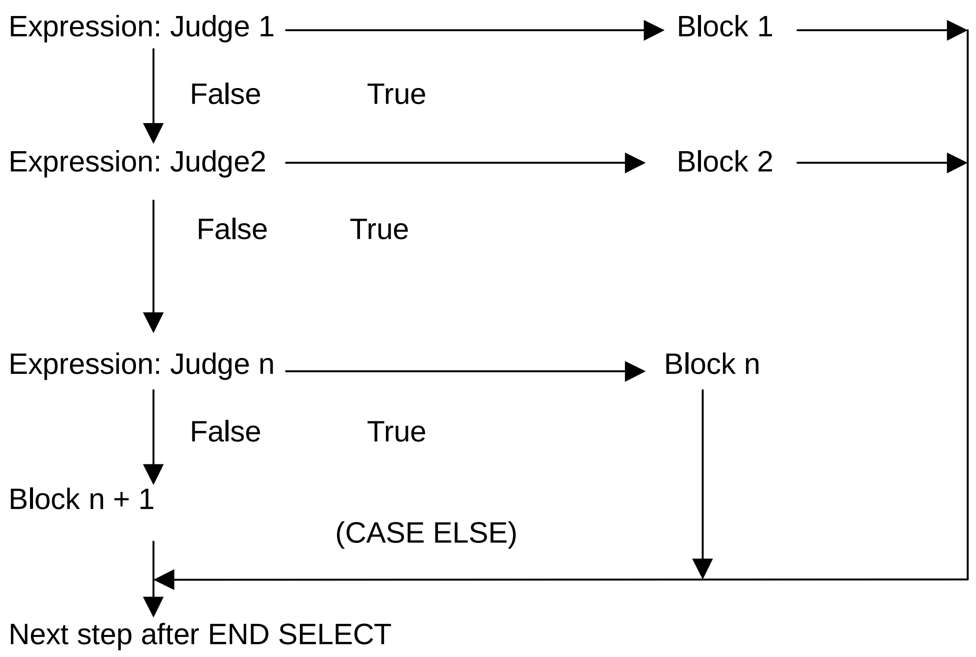
(2)

```
SELECT CASE A$
CASE "ABC","EF","FG" TO B$,IS < "ZZZ"
                                  If A$ is from "ABC", "EF" and "FG" to B$
<Block 1>                         or less than "ZZZ", Block 1 is executed.
                                        (Judge 1)
 CASE C$ + D$
                                        If A$ matches with C$+D$, Block 2 is
 <Block 2>                              executed. (Judge 2)

 CASE ELSE                              If Judge 1 and 2 do not match, no process
 END SELECT                             is executed.
```

[Explanation]

SELECT CASE block is processed as the flow below.



In SELECT CASE sentence, you can describe arbitrary numeric expression. At the beginning of SELECT CASE block, always start with SELECT CASE sentence.

For CASE sentence, describe <Judge expression> for <Expression> which is described in SELECT CASE. As the separator for <Judge expression>, ", " is used to describe several judgements below.

- Arbitrary numeric value, expression or character string
  If the result matches with the value of <Expression> in SELECT CASE, the block right after it is executed.

- <Expression 1>_TO_<Expression 2>
  <Expression 1> and <Expression 2> are arbitrary numeric value, expression or character string. If the value of <Expression> in SELECT CASE is between the value in <Expression 1> and the value in <Expression 2>, the sentence right after it is executed.

- IS_<Comparison operator>_<Expression> "
  < ", " > ", " >= " or " <= " can be used as <Comparison operator>. <Expression> is arbitrary numeric value, expression or character string. The sentence above must be described at the end of the judge expression. Comparing the value of <Expression> in SELECT CASE sentence to the value of <Expression> above, if it is true, the program right after the sentence is executed.

Note:

1. CASE sentence is not necessary to be described in SELECT CASE block.

2. It is not allowed to describe any sentence between SELECT CASE and CASE.

3. Any sentence including SELECT CASE block can be described to <Block>. After executing <Block>, the program jumps to the next step after END SELECT. It is allowed even if there is no block sentence.

4. CASE ELSE is to describe the sentence when all CASE sentences are false. One CASE ELSE is required at every SELECT CASE block.

5. END SELECT is to terminate SELECT CASE block. END SELECT must be always described with SELECT CASE sentence in SELECT CASE block.

6. The restrictions of this command are:

- The maximum multiplication of SELECT CASE block (depth of the layer of SELECT CASE blocks in a SELECT CASE block) is 8.

- CASE sentence can be described 127 sentences at maximum including CASE ELSE in one layer.

7. SELECT CASE, CASE ELSE and END SELECT do not create intermediate codes and are not counted as the execution steps.

SEQ~SEQEND (command)

[Function]

Specifies the range of robot sequence program.

[Format]

SEQ_#<File number>
SEQEND_#<File number>

[Example]

```
SEQ #1
...
SEQEND #1
```

[Explanation]

The robot sequence is the function to control the input/output ports while the robot is moving.

<File number> must be one of the values 0 to 15. <File number> corresponded to the communication port (COM 0, 1, 2, 3 or 8) must be specified by OPEN command in advance.

[Example]

For example, while the robot is moving, the dual chuck (RDC) can be operated as below.

When using the gripper 1 at the position A and the gripper 2 at the position B, the dual head can be rotated while the Z Axis is going up from A to a to change the gripper from 1 to 2.

If the rotation of the dual head has not completed while the robot is moving from a to b, the robot stops moving at the position b. After the rotation has completed at the position b, the Z Axis goes down.

The I/O control at the motion of the robot reduces the cycle time of the robot operation and this control can not be available in MOVE command.

In SEQ~SEQEND command, MOVE command must be used with FINISH command.

<Motion control of robot in SEQ~SEQEND command>

(1) Z Axis does not go down only by MOVE command.

When a MOVE command is executed, the Z Axis goes up from
position A and the robot moves to the target position B.
However, if a condition has not satisfied during the motion, the
robot stops at the position b.

(2) The program execution advances to the next step after MOVE command.

Normally, the program execution of a job is terminated until the robot completes positioning to the target position after MOVE command. However, when MOVE command has executed in SEQ~SEQEND command, the program execution of the job advances to the next regardless the positioning of the robot.

(3) The Z Axis down motion is allowed only by FINISH command.

While the robot is moving from A to b, if FINISH command is executed, Z Axis goes down from b to B without stopping the robot motion.

If FINISH command is not executed, the robot waits at the position b until the command is executed.

<Sample program>

```
 SEQ #1
 MOVE #1,PTP,PM(B)
       ....
 (Process at 1)
       ...
 WAIT (REF(#1,STATUS9) AND &H1) =1  .......... within Z SAFETY ZONE?
       ...
 (Process at 2)
       ....
 FINISH #1
 WAIT (REF(#1,STATUS9) AND &H1) =0  .......... out of Z SAFETY ZONE?
       ...
(Process at 3)
      ...
 WAIT (REF(#1,STATUS9) AND &H2) =2  .......... positioning completion?
 SEQEND #1
```

Note: The program branch commands, such as GOTO, can not be used to get out and/ or in the programs between SEQ~SEQEND command.

The (1) part in the "not applicable example" below is within SEQ~SEQEND, but the target branch (2) is out of the robot sequence and the part (2) is not processed as the robot sequence. Because the part (2) is out of the robot sequence, FINISH command is not executed and the robot stops moving at the MOVE command in the part (2).

<Not applicable example>

```
 SEQ #1
    ...
GOSUB *PICK
    ...
 SEQEND #1

 *PICK
   ...
MOVE #1, PTP, PM1
   ...
MOVE #1, PTP, PM2
```

```
   ...
RETURN
```

In order to perform positioning by the robot sequence, the sub routine program must be in SEQ~SEQEND as below.

<Applicable example>

```
SEQ #1
   ...
*PICK
   ...
RETURN

SEQEND #1
```

The nesting of SEQ~SEQEND is not allowed.

SET (command)

[Function]

Sets the motion characteristic of the robot.

[Format]

SET_#<File number>,<Characteristics>[,<Characteristics>[,...]]

[Example]

SET #1,SPEED=80

[Explanation]

Set the data concerning to the motion characteristics of the robot which is connected to the communication port specified by <File number>. <File number> is expressed by one of the value from 0 to 15. <File number> corresponded to the communication port (COM 0,1,2,3 or 8) must be specified by OPEN command in advance.

Refer to **ACCEL, SPEED, PULLUP, ARCH,SLOWUP, SLOWDOWN, ZZONE,WEIGHT**

<Characteristics> shown below can be set.

Characteristics		**Format**	**Explanation**
PTP speed	all axes	SPEED-<Value>	<Value> or <Value of N Axis> is specified by the percentage to rated speed. (<Value>: 0 to 100) This is used in PTP, GATE, ARCH and PASS motion.
	each axis	SPEED=([<Value of 1st axis>],[<Value of 2nd axis>],[,,,1]	
CP speed	linear	LINEA_SPEED=<Value>	<Value> is specified for the tip speed of the robot in mm/sec.
PTP acceleration and deceleration speed	all axes	ACCEL=<Value>	<Value> or <Value or N axis> is specified by the percentage of maximum acceleration speed. (<Value>: 0 to 100) This is used in PTP, GATE, ARCH PASS motion
	each axis	ACCEL=([<Value of 1st axis>],[<value of 2nd axis>],[...])	
CP acceleration and deceleration speed (Acceleration and deceleration distance)		CP_ACCEL=<Value>	<Value> specifies the distance for the acceleration and deceleration in mm. (0 to 999)
Auto pull up		PULL_UP=<Value>	<Value> specifies the distance from Z-axis origin point in mm. Upper positions at the GATE or ARCH motion.
Arch motion		ARCH=(<Value1>,<Value2>)	<Value 1>: Starting position of Arch up motion. <Value 2>: End position of Arch down motion.
SLOW UP		SLOWUP=(<Value1>,<Value2>)	<Value 1> specifies the distance from target position in mm. Speed at UP and DOWN motion is specified in <Value 2>. <Value 2> is specified by the percentage to rated speed. (<Value 2>: 0 to 100)
SLOW DOWN		SLOWDOWN=(<Value1>,<Value2>)	
Work weight		WEIGHT=<Value>	<Value> is specified by kg unit. (<Value>: 0 to 99.9)
Z ZONE		ZZONE=<Value>	<Value> specifies the distance from the origin of Z Axis in mm. (lower end of Z Zone) While Z Axis coordinates is in Z Zone, the Z safety zone flag of STATUS 9 or HSTATUS 9 becomes 1.

SETPRIORITY (command)

[Function]

Set and change the priority of a job and returned with number of the old priority.

[Format]

Old-priority = SETPRIORITY("Job-name", priority level)

[Example]

OLD.PRIORITY% = SETPRIORITY("TESTJOB", 1)

[Explanation]

SETPRIORITY function can change the priority of the specified job and returns with the number of the old priority. The priority of a job has the level as 1 (the lowest) to 10 (the highest). After initialising of STC, the priority of each job is set as 1 and each job runs equally.
The priority of a job means as follows with example. Among jobs with the priority 1, if you increase the priority of one job to 10, this job can run 10 steps during other jobs running 1 step. In the same way, if the priority ratio of 4 jobs has changed to 10:8:6:4, the executing step ratio changes to 5:4:3:2. If the all priorities have the same number (for example 10: 10:10), each job runs equally since the executing step ratio is flat as 1: 1:1:

Typical usage of SETPRIORITY is as follows.

```
JOB NAME "TESTJOB"
 SAVE% = SETPRIORITY("TESTJOB", 10)

 [High priority procedure]
 SAVE% = SETPRIORITY("TESTJOB", SAVE%)
```

[Errors]

If the specified job can not be found or the number of priority is out of range, the run time error 1A (Incorrect usage of command or function) occurs.

Refer to **GETPRIORITY**

SETROBNO (Function)

[Function]

Sets a robot number for the robot communication of a current job.

[Format]

SETROBNO(<Numeric expression>)

[Example]

```
SETROBNO（2）                    ‘set robot number #2 for this job
```

[Explanation]

When communicating with multiple robots through one communication port, a robot number is needed for the purpose of selecting a robot.

For a standard HNC-1XX,2XX,3XX,544 type controller, a robot can communicate without a robot number. But in case of a special ROM that can communicate by 1 for N structure using RS422 communication, a robot needs a robot number for the communication.
For a standard HNC-580 series controller, the communication needs a robot number because the controller can contain internally the virtual robots up to four. (But the communication without a robot number is also allowed. In this case, a robot with the number #1 is automatically selected for the communication.)

After **SETROBNO** function sets the robot number, robot control commands such as **MOVE**, **REF**, etc. communicate with a robot that has the number specified by **SETROBNO**.

The robot number must be specified by the value that is set to [MAINTENANCE]-[MAINTENANCE DATA]-[STATION NO.] in S.G. data of the controller.

It is needed that **SETROBNO** is executed in each job.

When a robot control command communicates, the communication uses the robot number that is set by the last **SETROBNO** execution. **SETROBNO** does not concern to using COM port. Therefore, if multiple robots are controlled through two or three ports in one job, **SETROBNO** or **CLEARROBNO** must be executed every time before a robot control command is executed.
Example)

```
      OPEN “COM1...” AS #1
      OPEN “COM2...” AS #2
      OPEN “COM3...” AS #3
                .
*LOOP
                .
      SETROBNO(2)
      MOVE #1,PTP,PM100            ‘COM1 robot no.#1
                .
      SETROBNO(3)
      MOVE #2,PTP,PM123            ‘COM2 robot no.#3
                .
```

```
CLEARROBNO( )
MOVE #3,PTP,PM214             ‘COM3 without robot no.
GOTO *LOOP
```

After STC power reset or program downloaded, a robot number of all jobs is set to the state as “without robot number”.

[Errors]

(Compiling error)

- If the specified robot number is the numerical constant with the value out of 0 – 999, the error “Illegal value of argument” occurs.

(Execution error)

- If the specified robot number has the value out of 0 – 999, the error “Incorrect usage of command or function” occurs.

Refer to **CLEARROBNO**, **GETROBNO**.

Note) **SETROBNO** is available for the following version.

- STC
  Available ROM version is 5.41 or later.
  In case of older version, an error occurs when downloading the program.

- HARL-III Compiler for DOS
  Can not compile in any versions. An error occurs when compiling.

- HARL-III Compiler for Windows
  Available version is 2.01 or later.
  In case of older version, an error occurs when compiling.

SGN (function)

[Function]

Gets the value indicating the sign of a number.

[Format]

SGN(<Numerical expression>)

[Example]

```
A% = SGN(B!/3)
B% = SGN(-10.34)      '-1 substituted for B%
C% = SGN(0.0)         '0 substituted for C%
D% = SGN(3.68)        '1 substituted for D%
```

[Explanation]

SGN returns the value indicating the sign of <Numeric expression>.
If <Numeric expression> has a negative value, **SGN** returns –1.
If <Numeric expression> has a zero value, **SGN** returns 0.
If <Numeric expression> has a positive value, **SGN** returns 1.

SIN (function)

[Function]

Gets the value of sine.

[Format]

SIN(<Numeric expression>)

[Example]

```
X=10*SIN(ANGLE)
```

[Explanation]

Gets the value of sine to the value of <Numeric expression> in radians. When the <Numeric expression> includes a double precision real number, the value is returned in a double precision. In other case, the value is returned in a single precision.

Refer to **ATN, COS, TAN**.

SLOWDOWN, SLOWUP (motion parameter memory)

[Function]

These memory define the values for insert motion and slow start motion

[Format]

SLOWDOWN=(<Value1>,<Value2>)	<Value 1 > :	distance to target position in mm
	< Value 2 > :	speed within slow down area in %
SLOWUP=(<Value1>,<Value2>)	<Value 1 > :	distance from target position in mm
	< Value 2 > :	speed within slow up area in %

[Example]

```
SET #1,SLOWDOWN=3,20

SET #8,SLOWUP=30,50
```

[Explanation]

When the power of the controller is turned on the values of SLOWDOWN and SLOWUP which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group). If later a SET command is executed by the HARL program the values will be overwritten.
<Value 1> specifies the distance to/from target position in mm.
Speed at UP and DOWN motion is specified in <Value 2>. <Value 2> is specified by the percentage to rated speed. (<Value 2>: 0 to 100)

Refer to **SET**

SM (motion parameter memory)

[Function]

This memory stores the S code of a position memory PM.

[Format]

SMm m : 0 - 999 The value of SM can be 00 to 99

[Example]

S%=REF (#1,SM2)

[Explanation]

In order to get the stored S data of a position memory PM you can read it with the REF command. It is not possible to write back the M code to the PM directly by this memory.

SM: SM is a S data stored in the robot controller. This memory can store the value from 0 through 99.

The SM data can be substituted by REF command.

Refer to **PM, REF**

SPACE$ (function)

[Function]

Gets the specified length of spaces.

[Format]

SPACE$(<Numeric expression>)

[Example]

```
B$=A$+SPACE$(12)+"ABC"
```

[Explanation]

Gets the string consisting of the spaces (character code &H20) specified in <Numeric expression>. The value of <Numeric expression> must be 0 to 255.

SPEED (motion parameter memory)

[Function]

This memory define the speed of all or several axes during automatic motion.

[Format]

SPEED=<Value>	< value > = 0 to 100 in %
SPEED=([<Value of 1st axis>],[<Value of 2nd axis>],[,,,N]	<Value> = 0 to 100 in %

[Example]

```
SET #1,SPEED=80

SET #1,SPEED=100,80,20
```

[Explanation]

When the power of the controller is turned on the value of SPEED which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group). If later a SET command is executed by the HARL program the values will be overwritten.

<Value> or <Value of N Axis> is specified by the percentage to rated speed. (<Value>: 0 to 100)

This is used in PTP, GATE, ARCH and PASS motion.

Refer to **SET**

SQR (function)

[Function]

Gets the square root.

[Format]

SQR(<Numeric expression>)

[Example]

```
A=SQR(X*X+Y*Y)
```

[Explanation]

Gets the square root of <Numeric expression> value. When <Numeric expression> includes a double precision real number, the double precision value is returned. In other case, the single precision value is returned.

STATUS (reserved variable)

[Function]

These variables content information of the condition of the robot controller section.

[Format]

STATUSm m : 0 - 9

[Example]

STATUS data can be retrieved by REF command like the example below.

```
* LOOP

IF (REF (#1, STATUS8) AND &H1) <> &H1 THEN GOTO *LOOP
(If the mode on the Teach Pendant is not in ON-LINE, the program execution is looped.)

MD0 = REF (#1, STATUS9) (Substitutes the value of STATUS9 for MD0.)

IF (MD0 AND &H4) = &H4 THEN GOTO *CALIB. OK
(If A-CAL has been completed, jump the program execution to CALIB. OK.)
```

[Explanation]

There are 10 memories (read only) from STATUS 0 through 9 (STATUS7 is not used) in order to show the conditions of a robot such as status data or error data. The status data always keeps the latest information. Each STATUS can be read by REF command.

<STATUS 0>

STATUS 0 shows one of the error codes (hexadecimal) below.

Error code	**Description**
&H00	normal condition
&H10	Emergency stop condition
&H11	Deadman switch is on.
&H20	A-CAL incomplete (See STATUS 1 to 6)
&H21	Positioning does not complete.
&H30	Address is out of limit.
&H31	M data is not normal.
&H32	W Axis Sensor Stop does not work.
&H33	Data for free curve movement is not normal.
&H34	EPI retry error
&H40	Position data is out of limit area (See STATUS 1 to 6)
&H51	Overrun of robot (See STATUS 1 to 6)
&H60	Communication format error
&H61	Communication command error
&H62	Received a command which is not available in the mode.
&H63	System data (SG, SP) are destroyed.
&H64	Position data are destroyed.
&H65	ON-LINE communication error
&H66	Watch dog time out error *
&H72	Servo error (See STATUS 1 to 6.)
&H80	Received a command while executing another command.
&H90	Moving distance is too short.
&H91	Pass motion PTP overflow error
&H92	Pass motion PTP underflow error
&H93	Over speed error
&H95	Coordinates conversion error
&H96	Final positioning can not be completed.
&H99	Motor does not work.
&HA0	Abnormal status at servo driver
&HB0	Abnormal status at encoder line

<STATUS 1 to 6>

STATUS 1 to 6 store the error data of each axis. Those memories are applicable when the error code of STATUS0 is related to the robot motion. (Error code &H20, &H40, &H51 or &H72 in Table 2.6.)

STATUS 1	Error data of X axis
STATUS 2	Error data of Y axis
STATUS 3	Error data of Z axis
STATUS 4	Error data of W axis
STATUS 5	Error data of 5th axis
STATUS 6	Error data of 6th axis

(1) When the error code of STATUS0 is &H20, A-CAL error is stored by the value 0 to 7 as error data of each axis to STATUS 1 to 6. (Refer to operation manual of the robot.)

(2) When the error code of STATUS0 is &H40, the area error is stored by the value 0 to 2 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Lower side, 2: Upper side)

(3) When the error code of STATUS0 is &H51, overrun is stored by the value 0 to 3 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Origin, 2: Overrun side, 3: Both)

(4) When the error code of STATUS0 is &H72, servo error is stored by the value 0 to 1 as error data of each axis to STATUS 1 to 6. (0: Normal, 1: Error)

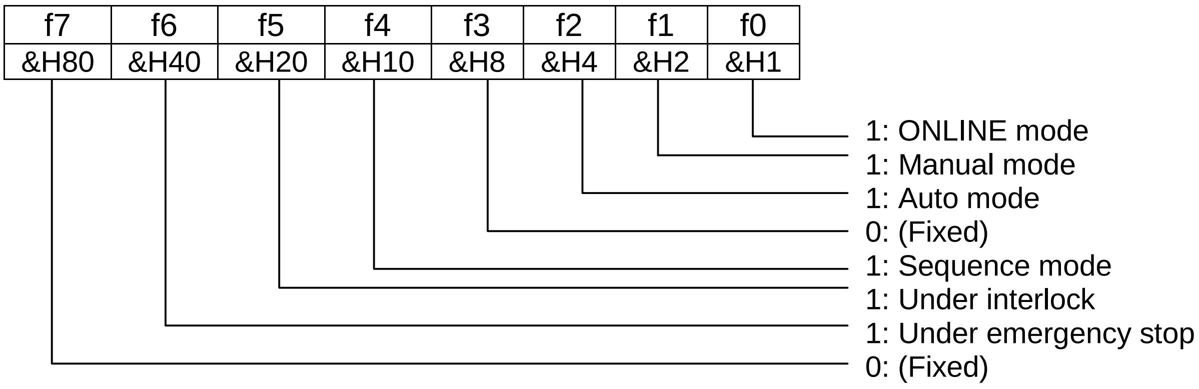
For example, if the error code of STATUS0 is &H51, one or some of the robot axes is in overrun status. When the data of STATUS2 is 1 at that time, Y Axis is in overrun at its origin side.

STATUS 8, 9 consist of 8 bit flags, and these flags are assigned to check the mode or status of the robot.

Each flag has the "weight ," and the total amount of flag weight which are turned on (1) is the value of STATUS m.

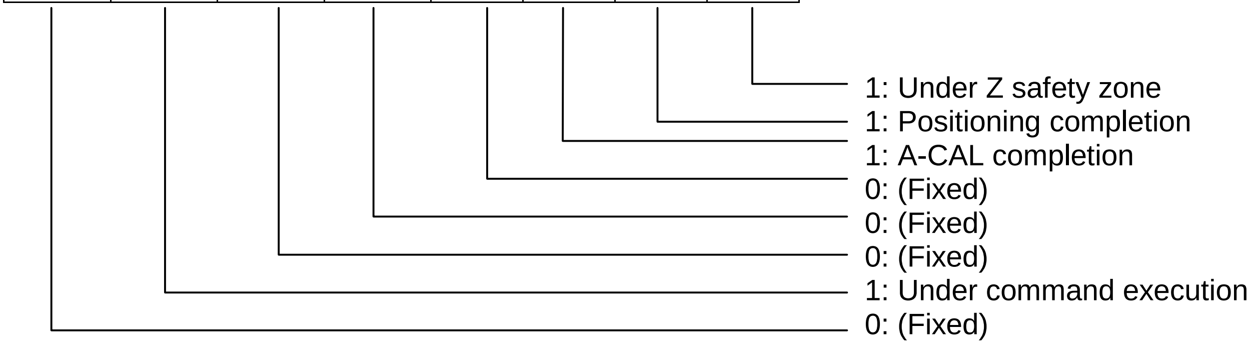
<STATUS 8>

f7	f6	f5	f4	f3	f2	f1	f0
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1



<STATUS 9>

f7	f6	f5	f4	f3	f2	f1	f0
&H80	&H40	&H20	&H10	&H8	&H4	&H2	&H1



Before using REF command to read STATUS, the communication port should be opened by OPEN "COM command in advance.

Refer to **REF**

STR$ (function)

[Function]

Converts the value to character string.

[Format]

STR$(<Numeric expression>)

[Example]

A$=STR$(NUM)+NAME$

[Explanation]

Converts the value of <Numeric expression> to character string. If the value of <Numeric expression> is plus, the first character of the string is started with a space after the conversion. If the value of <Numeric expression> is minus, the first of the string is started with " - " (minus symbol) after the conversion.

Note: Pay attention to the difference between STRING$ and STR$.

Refer to **HEX$, STRING$, VAL**.

STRING$ (function)

[Function]

Gets the character strings connecting the specified numbers of the character.

[Format]

STRING$(<Numeric expression 1 >,<Character expression>) or <Numeric expression 2>

[Example]

A$=STRING$(50,"+")

[Explanation]

Gets the numbers of character string specified in <Numeric expression 1> connecting the characters specified in <Character expression> or corresponded to character code specified in <Numeric expression 2>. When specifying it by <Character expression>, only the first character is used. The specification by <Numeric expression 2> must be 0 to255.

Refer to **STR$**.

TAN (function)

[Function]

Gets the value of tangent.

[Format]

TAN(<Numeric expression>)

[Example]

```
A=TAN(123)
```

[Explanation]

Gets the value of tangent to the value of <Numeric expression>. The unit of <Numeric expression> is in radian. When <Numeric expression> includes double precision real number, the value is returned in a double precision. In other case, the value returned in a single precision.

Refer to **ATN, COS, SIN**.

TIM (variable)

[Function]

The timer starts upon executing when it is set by LET or "=" command. The counting timer is not interrupted by JOB OFF command.

[Format]

TIMm = <value>	m : 0 -31	<value> : 0.00 through 327.67 seconds. The value 0 is converted to - 1 immediately.)

[Example]

recommended usage:

```
TIM2=2.00
IF TIM3 THEN GOTO * TIMEUP              (After TIM3 is time up, jump to * TIMEUP.)
```

not recommended usage

```
TIM2=2.00
IF TIM3 = 0 THEN GOTO * TIMEUP        (Because there is not instant at time 0, the program is
                                       not executed correctly.
```

[Explanation]

As soon as the times is set by LET command or equal sign it starts to count back to zero while the program execution goes on.
Any timer can be used at each job. However, if a new value is set to a timer which is counting down, the value of the timer is rewritten to the new value.
The condition of time Up can be used by IF sentence. In this case, time up condition is true.

Note: In HARL-III, the value "- 1" is regarded as true. However, there is no value "-1" in timer. In order to describe the time up condition as true, the value is converted to - 1 when the value becomes 0. Therefore, in case of example 2, the timer can not work properly. Express as example 1 below.

Refer to **IF, WAIT**

TIME$ (reserved variable)

[Function] Sets or retrieves the time of internal clock.

[Format]

(1) TIME$

(2) TIME$=" hh:mm:ss"

[Example]

(1) A$=TIME$

The current time is substituted for A$.

(2) TIME$=" 12:34:56"

Sets the time 12 hour 34 minute 56 second.

[Explanation]

The current time is always set in TIME$ of the form "hh:mm:ss" and always can be retrieved.

hh (Hour)	00 to 23
mm (Minutes)	00 to 59
ss (Seconds)	00 to 59

When the time of TIME$ becomes "00:00:00", the date of DATE$ changes.

Note: The value of TIME$ is a character string data.

Refer to **DATE$.**

TIMEOUT (reserved variable)

[Function]

Keeps the time out status used by WAIT command.

[Format]

TIMEOUT

[Example]

IF TIMEOUT THEN GOTO *T.OUT

[Explanation]

If the WATT command executed at latest has time out, true (- 1) is kept. If it has not time out, false (0) is kept. The status is kept until the next WAIT command is executed.

One TIMEOUT variable can be used at each job and the variable is related to the execution of WAIT command in the job. TIMEOUT variable is to refer the execution if WAIT command and no value is substituted.

The false (0) is set to TIMEOUT variable initially.

Refer to **WAIT.**

VAL (function)

[Function]

Converts the numeric value of the character string to the real value.

[Format]

VAL(<Character string>)

[Example]

A=VAL(A$)

[Explanation]

Returns the numeric value converting the value expressed by <Character string>. <Character string> can be specified by the integer type (decimal number or hexadecimal number) and the real number type. If the first character of <Character string> is not " + ", " - ", " & " or number, the value of VAL is 0. If the character excluding numeric value contained in <Character string>, the characters after it are ignored. However, in case of hexadecimal number, A to F are regarded as the number. The space in character string are ignored.

Refer to **STR$.**

WAIT (command)

[Function]

Halt the program execution until the specified conditions or time has satisfied.

[Format]

WAIT_<Logical expression>[,<Numeric expression>]

[Example]

```
WAIT INB5=1
WAIT INB3=0,8
```

Wait until the input port (INB3) becomes off. The maximum time to wait it is8 seconds.

[Explanation]

If the condition specified by <Logical expression> is true (- 1), the program advances to the next step. If it is false (except for - 1), the program execution is halted at the line until the condition has been satisfied.

Only the following expressions are allowed as <Logical expression>.

• <Expression> <Relational operator> <Expression>
  Note) Relational operator : <, >, =, <>, <=, <=
• **EOF** or **NOT EOF**

<Numeric expression> is to restrict the wait time and its set range is 0 to 327.67 seconds. When the set time has passed, even if the condition specified by <Logical expression> is not satisfied, the execution of WAIT command is terminated and the true is set to TIMEOUT variable which keeps whether or not WAIT command has time out.

Therefore, the next step of WAIT command is programmed as

```
IF TIMEOUT THEN GOTO *T.OUT
```

Then, the program can be jumped if the condition set by **WAIT** command has not satisfied in the specified time.

If <Numeric expression> is omitted, the program execution is halted until the specified condition is satisfied.

This command is valid to the job executed only.

Note: If **JOB OFF** command is executed while **WAIT** command is working, the wait status is resumed after **JOB ON** command. The timer counting for **TIMEOUT** is stopped during **JOB OFF** status.

Refer to **TIMEOUT**.

WEIGHT (motion parameter memory)

[Function]

This memory defines the value of work weight which is used to calculated optimized acceleration and deceleration.

[Format]

WEIGHT=<Value>   < Value > = 0.00 to 99.9 in kg

[Example]

SET #1,WEIGHT=4.00

[Explanation]

When the power of the controller is turned on the value of WEIGHT which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group). If later a SET command is executed by the HARL program the values will be overwritten.
<Value> is specified by kg unit. (<Value>: 0 to 99.9)

Refer to **SET**

XOR (logical operator)

[Function]

Logical operation is used for checking several conditions, bit operation or pool operation.
Logical operation gives 0 or 1 to each bit as result.
This logical operator is exclusive logical addition.

[Format]

< Variable > XOR < Variable >

[Example]

1)	12 XOR 11 = 7	12 = (1100)2, 11 = (1011)2 Therefore, 12 XOR 11 = (0111)2 = 7
2)	10 XOR 10 = 0	10 = $(1010)_2$ Therefore, 10 XOR10 = (0000)2 = 0

[Explanation]

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

Logical operation is also used for changing program flow as relational operator. In this case, logical operator can be connected by one relational operator and more.

Refer to **AND, OR**

ZZONE (motion parameter memory)

[Function]

This memory defines the value of the safety zone.

[Format]

ZZONE=<Value>   < Value > = 0.00 to 999.99 in mm

[Example]

SET #1, ZZONE=150

[Explanation]

When the power of the controller is turned on the value of SAFETY ZONE which is stored in the SYSTEM PARAMETER is valid and copied to the operation register (REMOTE group). If later a SET command is executed by the HARL program the values will be overwritten.
<Value> specifies the distance from the origin of Z Axis in mm. (lower end of Z Zone)
While Z Axis coordinates is in Z Zone, the Z safety zone flag of STATUS 9 becomes 1.

Note : The value of safety zone have to be larger than of pull up. Otherwise the Z safety zone flag will not be switched on. Also this function doesn't work as long as the robot isn't calibrated.

Refer to **SET**

5. APPENDIX

5.1 Character Code

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	I	y
A	LF	SUB	*	:	J	Z	j	z
B	HM	ESC	+	;	K	[	k	{
C	CL	→	,	<	L	¥	l	
D	CR	←	-	=	M	]	m	}
E	SO	↑	.	>	N	^	n	~
F	SI	↓	/	?	O	_	o	

5.2 Reserved Word (Can not be used for variable and label name)

ABS	GOTO	NOISE	SPLINE
ACCEL	HERE	NOT	SQR
AND	HEX$	OFF	START
ARC	HINB	ON	STATUS
ARCH	HIND	OPEN	STEP
ARM	HOLD	OR	STOP
AS	IF	ORB	STR$
ASC	INB	ORD	STRING$
ATN	INCLUDE	OUTB	TAN
CALIB	IND	OUTD	THEN
CAMERA	INPUT	P	TIM
CASE	INPUT$	PAI	TIME$
CHR$	INSTR	PASS	TIMEOUT
CIRCLE	INT	PICTURE	TO
CLEARROBNO	INVERT	PM	TOOL
CLOSE	IRB	PRINT	VAL
COM	IRD	PTP	VAR
COMPLIANCE	JOB	PULSE	VCALIB
COS	JOG	PW	VDA
CP	LEFT$	PX	VDX
DATE$	LEFTY	PY	VDY
DELAY	LEN	PZ	VGA
DIM	LET	REF	VGX
DIMNET	LEVEL	REM	VGY
DIMPOS	LINE	RESPONSE	VMODE
DISABLE	LINEAR	RESUME	VRESULT
DRIVE	LOCAL	RETURN	VSET
ELSE	LOG	RIGHT$	WAIT
END	MB	RIGHTY	WEIGHT
ENDIF	MD	RND	XOR
EOF	MEASURE	ROTARY	ZUP
ERR	MID$	SELECT	ZZONE
ERROR	MM	SEQ	
EVENT	MOD	SEQEND	
EXP	MODE	SET	
EXTRACT	MOVE	SETPRIORITY	
FINISH	MRB	SETROBNO	
FIX	MRD	SGN	
FM	MW	SIN	
FOR	NAME	SLOW	
GATE	NETCLOSE	SLOWDOWN	
GETPRIORITY	NETOPEN	SLOWUP	
GETROBNO	NETREAD	SM	
GLOBAL	NETWRITE	SPACE$	
GOSUB	NEXT	SPEED	

5.3 Error Code List (Errors at the program execution)

Error code	Content
01 (01)	Program pointer is out of order
02 (02)	Bus Error Exception
03 (03)	Address Error Exception
04 (04)	Illegal Instruction Exception
05 (05)	Zero Divide Exception
06 (06)	CHK Instruction Exception
07 (07)	TRAPV Instruction Exception
08 (08)	Privilege Violation
09 (09)	Format Error
0A (10)	Line 1010 Emulator Exception
0B (11)	Line 1111 Emulator Exception
0E (14)	Communication error between ALU and CPU

The errors 01 through 0E are the internal system errors If the error above happens, please contact Hirata distributor.

Error code	Content
0F (15)	Program has destroyed. Download your program again.
10 (16)	Overflow of operation result
11 (17)	The operation by 0 was executed
12 (18)	RESUME was executed without the error occurence
13 (19)	Not used.
14 (20)	Not used.
15 (21)	The array variable was used beyond the range specified by DIM.
16 (22)	The nests of FOR~NEXT were more than 8.
17 (23)	FOR and NEXT were not used in a pair.
18 (24)	A command that is not applicable was used.
19 (25)	RETURN was executed without GOSUB.
1A (26)	The usage of command or function is not correct.
1B (27)	The communication port (COM number) which had already opened was attempted to open.
1C (28)	Attempted to use an unopened file.
1D (29)	There is no data to be read in a file.
1E (30)	Not used.
1F (31)	Overflow the buffer (255 characters) for data receipt.
20 (32)	Stack error while executing FOR~NEXT command.
21 (33)	The expression of character string was too complex.
22 (34)	The character string that is more than 255 characters was attempted to substitute for character type variable.
23 (35)	Type of variable did not match.
24 (36)	The command did not match the correct format.
25 (37)	Not used.
26 (38)	Co-processor was out of order.
27 (39)	The error occurs at data receipt.

28 (40)	Attempted to open the file number (#n) which had already used.
29 (41)	Stack error
2A (42)	The nests of GOSUB~RETURN were more than 8.
2B (43)	Communication error due to writing problem.
2C (44)	Communication error due to overflow of communication buffer.
2D (45)	Attempted to execute JOB START a job which has not in JOB OFF state.
2E (46)	The position memory P is out of the range specified by DIMPOS.
2F (47)	Attempted to open the network communication twice.
30 (48)	Overflow the number of opened network communications
31 (49)	Specified network communication has not been opened.
32 (50)	Illegal size of data when writing to a network communication.
33 (51)	CR (Communication Reference) is not defined for specified network communication.
34 (52)	Own station number is specified.
50 (80)	The communication with the robot controller is impossible.
51 (81)	Robot error. The robot controller sent error code to STP.
52 (82)	The reply from the robot controller is not defined one.
53 (83)	The coordinates or M data are not set in the robot memory.
54 (84)	The robot is not in SEQ mode.
55 (85)	Virtual robots to send multiply overflow
56 (86)	Duplicated sending to robots
57 (87)	Received position data invalid
58 (88)	Robot response without robot no.
59 (89)	Different robot no. in response
60 (96)	OPEN was executed in through mode.
61 (97)	CLOSE was executed in through mode.

5.4 Error Message (checked at compilation)

No.	Message	Meaning
0	Syntax error	A sentence is not described in accordance with the syntax.
1	Division by Zero	Division by zero was attempted.
2	Duplicated label	The same label name is described in a job.
3	FOR without NEXT	A FOR was encountered without a matching NEXT. (The number of FORs is greater than that of NEXTs.)
4	Illegal function call	The way of calling function is wrong.
5	Line buffer overflow	Attempted to enter data over the allowed range of a line.
6	NEXT without FOR	A NEXT was encountered without a matching FOR. (The number of NEXTs is greater than that of FORs.)
7	Out of memory	The available memory area becomes short.
8	IF formula too complex	IF sentence is too complex.
9	Overflow	The operation result or the entered value is out of the allowed range.
10	RETURN without GOSUB	A RETURN was encountered without matching GOSUB. (The number of RETURNs is greater than that of GOSUBs.)
11	String formula too complex	The character expression is too complex. (too many nestings with parentheses etc)
12	String too long	The number of the characters in a character variable is greater than 128 characters.
13	Subscript out of range	A subscript of an array element is outsize the dimension of the array.
14	Type mismatch	The left side of the expression does not match the right side, or the variable type is different.
15	Undefined label	A label that was not defined is used.
16	Undefined line number	The line number to be branched was not defined.
17	Duplicate variable	The usage of variable is wrong.
18	Variable name too long	A variable name is not within 16 characters.
19	Label name too long	A label name is not within 16 characters.
20	FOR statement missing	The format of FOR description is wrong.
21	IF statement missing	The format of IF description is wrong.
22	Misplace ELSE	An ELSE does not match an IF sentence.
23	GOTO statement missing label	The label after GOTO sentence is wrong.
24	Too much code defined in file	The contents described in a file are too complex.
25	Too much code defined in line	The contents described in a line are too complex.
26	Duplicate definition	The array or user function is defined in duplicate.
27	Too much dimension number	The dimension of the array is greater than 3.
28	Constant expression required	The array dimension was not declared by constant.
29	Undefined JOB NAME statement	There is no JOB NAME command at the begging of the job program.

30	Illegal octal digit	A character that is not valid is described in octal constant.
31	Illegal hex digit	A character that is not valid is described in hexadecimal constant.
32	Too many decimal points	There are several decimal points in a floating-point constant.
33	DIM statement missing	The array dimension was not declared by DIM command.
34	Invalid indirection	The array was not declared.
35	DIMPOS statement missing	The value declared by DIMPOS is out of the allowed range.
36	EOF number out of range	The file number of EOF is out of the allowed range.
37	Bad file mode	The designation of file mode in OPEN command is wrong.
38	File number out of range	The value of file number is out of the allowed range.
39	DELAY statement missing	The description of DELAY is wrong.
40	Port number out of range	The port number is out of the allowed range.
41	ERROR statement missing	The description of ERROR is wrong.
42	LINE INPUT statement missing	The description of LINE INPUT is wrong.
43	MACRO statement missing	The description of MACRO is wrong.
44	Not found INCLUDE file	The header file described in INCLUDE statement is not found.
45	Specified axis missing	The description of axis designation is wrong.
46	MOVE statement missing	The description of MOVE is wrong.
47	Position count mismatch	The number of position data does not match that of motion pattern.
48	Illegal precious number	The value of precision is out of the allowed range.
49	Option missing	The description o options in MOVE or SET command is wrong.
50	Component expression missing	The description of position data components is wrong.
51	Duplicate SEQ statement	SEQ statement is defined in duplicate.
52	SEQEND without SEQ	A SEQEND was encountered without a matching SEQ. (The number of SEQENDs is greater than that of SEQs.)
53	DRIVE statement missing	The description of DRIVE is wrong.
54	JOG statement missing	The description of JOG is wrong.
55	OPEN statement missing	The description of OPEN is wrong.
56	Overflow in numeric constant	The numeric constant is too large.
57	SEQ without SEQEND	A SEQ was encountered without a matching SEQEND. (The number of SEQs is greater than that of SEQENDs.)
58	WHILE without WEND	A WHILE was encountered without a matching WEND. (The number of WHILEs is greater than

		that of WENDs.)
59	WEND without WHILE	A WEND was encountered without a matching WHILE. (The number of WENDs are greater than that of WHILEs.)
60	Not enough memory	The memory required for the system is not enough.
61	DEFINE statement missing	The description of DEFINE is wrong.
62	Duplicate JOB NAME	The JOB NAMEs are defined in duplicate.
63	Undefined JOB NAME	The JOB NAME is not found.
64	Bad object file	The object file is in an abnormal state.
65	Bad label file	The label file is in an abnormal state.
66	Bad local variables file	The local variable file is in an abnormal state.
67	Bad global variables file	The global variable file is in an abnormal state.
68	More than 32 jobs defined	The number of defined jobs is greater than 32.
69	Too many global variables	The number of global variables is more than 300.
70	Bad make file	The make file is in an abnormal state.
71	ENDIF without IF	The number of IF statements is more than that of ENDIFs.
72	SELECT CASE nesting over	The nesting for SELECT CASE is more than 8.
73	SELECT CASE statement missing	There is no SELECT CASE statement.
74	IS operator is not tail	IS operator is not set at the end.
75	CASE ELSE statement missing	There is no CASE ELSE statement.
76	END SELECT statement missing	There is no END SELECT statement.
77	Sequence error in SELECT CASE block	The sequence in SELECT CASE block is not normal.
78	CASE statements more than 127	The number of CASE statements is more than 127.
79	Statement between SELECT CASE and CASE	There is an executable statement between SELECT CASE and CASE statements.
80	WAIT expression not logical	The description of WAIT sentence is not correct.
81	Illegal assignment	The substitution is not allowed.
82	Illegal robot type	The robot type is wrong.
83	Bad argument type of function	An argument type for a function is wrong.
84	Illegal value of argument	An argument value for a function is wrong.

5.5 Error Code Stored in STATUS0

Error Code	Description
&H00	Normal condition
&H10	Emergency stop condition
&H11	Dead-man switch is on.
&H20	A-CAL incomplete (See STATUS1 to 6)
&H21	Positioning does not complete.
&H30	Address is out of limit.
&H31	M data is nor normal.
&H32	W Axis Sensor Stop does not work.
&H33	Data for free curve movement is not normal.
&H34	EPI retry error
&H40	Position data is out of limit area. (See STATUS1 to 6)
&H51	Overrun of robot (See STATUS1 to 6)
&H60	Communication format error
&H61	Communication command error
&H62	Received a command that is not available in the mode.
&H63	System data (SG, SP) are destroyed.
&H64	Position data is destroyed.
&H65	ON-LINE communication error
&H66	Watch-dog time out error *
&H70	Servo error (See STATUS1 to 6)
&H80	Received a command while executing another command.
&H90	Moving distance is too short.
&H91	Pass motion PTP overflow error.
&H92	Pass motion PTP underflow error
&H93	Over speed error
&H94	M number is not proper.
&H95	Coordinates conversion error
&H96	Final positioning can not be completed.
&H99	Motor does not work.
&HA0	Abnormal status at servo driver
&HB0	Abnormal status at encoder line

* This error will not happen at some controllers.