Fuji Electric

Innovating Energy Technology

MICREX-SX *series*

SPH

USER'S MANUAL

Data Linkage

<Applicable Models>

ISA bus adapted high-performance CPU board (type: NP3PS-SX1PCS-32)
ISA bus adapted slave board (type: NP3L-SX1SASS)
PCI bus adapted high-performance CPU board (type: NP3PS-SX1PCS-32, NP3PS-SX1PCS-74)

FEH244a

Preface

Thank you for purchasing Fuji Electric Programmable Controller MICREX-SX Series.
This User's Manual <Data Linkage> explains the method for reading or writing the internal data of an ISA or PCI bus adapted CPU board or of the CPU that is connected to an ISA bus adapted SX bus slave board from an ISV (Independent software vendor) application program.
Read this manual carefully to ensure correct operation. When using modules or peripheral devices, be sure to read the corresponding user's manuals listed below:

Title	Manual No.	Contents
User's Manual, ISA bus adapted high-performance CPU board, MICREX-SX series	FEH235	Explains the hardware specifications of ISA bus adapted high-performance CPU board and the method for installing the driver software necessary to use it.
User's Manual, ISA bus adapted slave board, MICREX-SX series	FEH236	Explains the hardware specifications of ISA bus adapted slave board and the method for installing the driver software necessary to use it.
User's Manual Hardware, MICREX-SX series SPH	FEH201	Explains the system configuration, the hardware specifications and operations of modules in the MICREX-SX series.
User's Manual Instruction, MICREX-SX series	FEH200	Explains the memory, language and system definitions of the MICREX-SX series.
User's Manual D300win <Reference>, MICREX-SX series	FEH257	Explains the installation procedure, functions and operations of D300winV3.
User's Manual, PCI bus adapted highperformance CPU board, MICREX-SX series	FEH242	Explains the hardware specifications of PCI bus adapted high-performance CPU board and the method for installing the driver software necessary to use it.
User's Manual SX-Programmer Standard <Instruction>, MICREX-SX series SPH	FEH588	Explains the memory, instructions and system definitions of the SPH series in the case of using SX-Programmer Standard.
User's Manual SX-Programmer Standard <Reference>, MICREX-SX series SPH	FEH590	Explains the functions and the operations of SX-Programmer Standard.
User's Manual, Driver for PCI Bus Adapted High-performance CPU board, MICREX-SX series	FEH245	Explains the specifications and operations of the driver for high-speed access to the memory in the PCI bus adapted high-performance CPU board.

In addition to the above manuals, the following Fuji Electric FA Components & Systems Co., Ltd. site offers various manuals and technical documents associated with MICREX-SX.
URL http://www.fujielectric.co.jp/fcs/eng/

Notes

1. This manual may not be reproduced in whole or part in any form without prior written approval by the manufacturer.
2. The contents of this manual (including specifications) are subject to change without prior notice.
3. If you find any ambiguous or incorrect descriptions in this manual, please write them down (along with the manual No. shown on the cover) and contact FUJI.

Revision

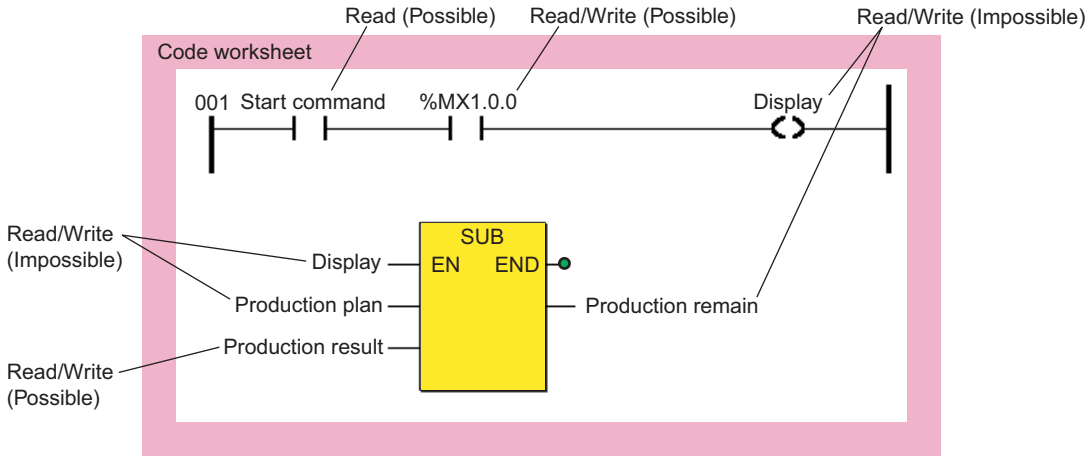
*The manual No. is printed at the bottom right of the cover of this manual.

Printed on	*Manual No.	Revision contents
Jul. 2002	FEH244	• First edition
Sep. 2006	FEH244a	• Some functions were added.

Contents

Section 1 Application

This manual explains how to read or write the internal data of an SX bus adapted high-performance CPU board or of the CPU that is connected to an SX bus slave board from an ISV application program.
To read or write CPU internal data, the data must be a variable that is given an AT address or a direct variable that is set a direct address from the code worksheet. No error, however, occurs even if a memory for which no address is set is read or written.



Variable worksheet

```
1
2   VAR
3           Start command       AT  %IX1.0.0  :   BOOL;
4           Display                           :   BOOL;
5           Production plan                   :   INT   := 3000;
6           Production result   AT  %MW3.     :   INT:
7           Production remain                 :   INT;
8   END_VAR
9
```

In the sample program shown above, AT addresses are set for two variables, or “Start command” and “Production result”, and “%MX1.0.0” is used as a direct variable on the code worksheet. These 3 variables can be accessed from ISV application program.
On the other hand, no AT address is set for other 3 variables, or “Display”, “Production plan” and “Production remain”, and therefore these variables cannot be accessed from ISV application program.

Section 2 Structure of Software Files

To make an ISV application program, the following files are necessary:

(1) Interface DLL (DCOMMIF.dll)

This file provides 8 types of data linkage function. For more information, refer to Section 4 (Provided Functions).

(2) Import library (DCOMMIF.lib)

This import library is linked when the interface DLL is used by load time dynamic linking.

(3) Included files (DCIDEF.h, dcidef2.h)

Export API and error code definition for data linkage function DLL

(4) Message manager related files (MsgMng.exe, DLL, initialization file)

Message manager is the software module that has the function to arbitrate between drivers and application programs when multiple application programs are accessing various types of communication board.
When the data linkage function is used, MsgMng.exe has to be started in advance.

(5) Others (software requirements)

<When ISA bus adapted high-performance CPU board or ISA bus adapted SX-bus slave board is used>

- WindowsNT Workstation Version 4.0 (Service Pack 3 or higher)
- TCP/IP protocol

<When PCI bus adapted high-performance CPU board is used>

- WindowsNT Workstation Version 4.0 (Service Pack 3 or higher)
- Windows2000 Professional (Service Pack 1 or higher)
- WindowsXP Professional (Service Pack 1 or higher)
- TCP/IP protocol

Section 3 How to Use

3-1 Installation

Copy the following files that are included in the floppy disk supplied with the product in specified directories.

1) DCOMM.dll (interface DLL)
   →WindowsNT directory \System32

2) dcidef2.h, DCIDEF.h (header file)
   →Specified directory (arbitrary) \ i nclude

3) DCOMMIF.lib (import library)
   →Specified directory (arbitrary) \lib

4) The following 7 message manager related files
   <When ISA bus adapted high-performance CPU board or ISA bus adapted SX bus slave board is used>
   MsgMng.exe, isaplc.dll, loaderIF.dll, Sxbus_s.dll. MsgMng081.dll, loader_if.ini. MsgMng.ini
   →Specified directory (arbitrary) \

   <When PCI bus adapted high-performance CPU board is used>
   MsgMng.exe, isaplc.dll, loaderIF.dll, Sxbus_s.dll. MsgMng001.dll, MsgMng081.dll, loader_if.ini, MsgMng.ini
   →Specified directory (arbitrary) \

<When PCI bus adapted high-performance CPU board is used on the OS of English version>

When PCI bus adapted high-performance CPU board is used on the OS of English version, it is necessary to modify the description of Msgmng.ini file.

```
[MsgMng]
COUNTRY = 081
    ↓
[MsgMng]
COUNTRY = 001
```

Note: Neither ISA bus adapted high-performance CPU board nor ISA bus adapted SX bus slave board is adapted to the OS of English version.

3-2 How to Make an Application Program

Including Header Files and Linking Import Library (Shown below is an example for Visual C++)

1) Include header files “dcidef2.h” and “DCIDEF.h” in the source. Include them, referring to the example shown below:
   (Example)
   • #include “Installation directory \include\dcidef2.h”, or
   • #include <dcidef2.h>
   In this case, inclusion path needs to be set from workspace (project), etc.

2) Set workspace (or project, make file or link parameter) so that import library DCOMMIF.LIB will be linked.

3) When this driver is upgraded, it is necessary to rebuild (recompile or re-link) it using new header files and import library.

Section 4 Provided Functions

DCOMMIF.dll provides the following functions:

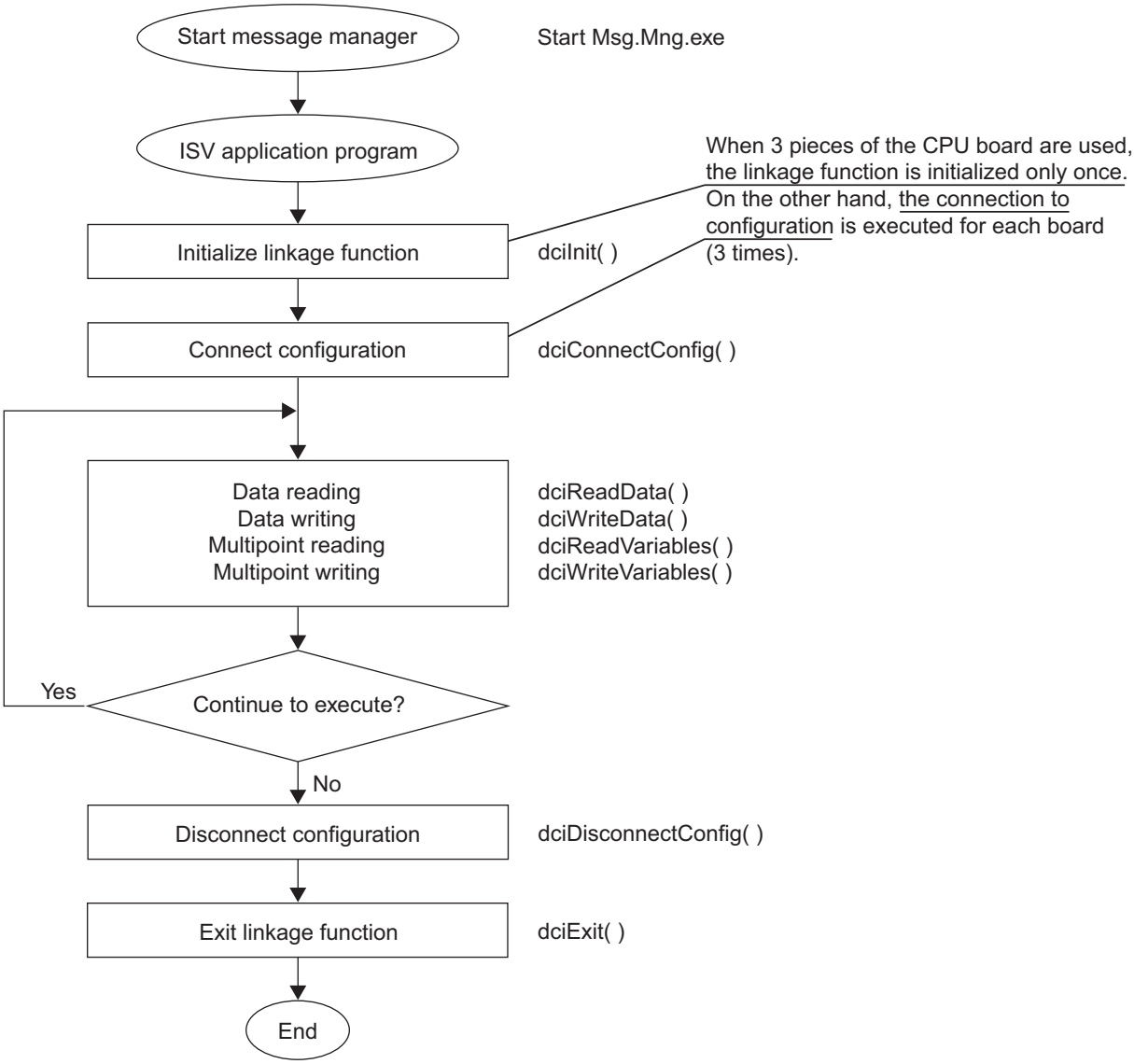
PCI bus adapted high-performance CPU board
ISA bus adapted high-performance CPU board
ISA bus adapted SX-bus slave board

* O: Available, X: Not available

No.	Name	Function name	Remark			
1	Initialize linkage function	dciInit()	Initialize the environment of the data linkage function	O	O	O
2	Exit linkage function	dciExit()	Release the environment of the data linkage function	O	O	O
3	Connect configuration	dciConnectConfig()	Establish the connection to a configuration	O	O	O
4	Disconnect configuration	dciDisConnectConfig()	Disconnect a configuration	O	O	O
5	Data reading	dciReadData()	Read data from specified address.	O	O	O
6	Data reading (Adapted to command status)	dciReadDataS()	Read data from specified address.	X	X	O
7	Data writing	dciWriteData()	Write data in specified address.	O	O	O
8	Data writing (Adapted to command status)	dciWriteDataS()	Write data in specified address.	X	X	O
9	Multipoint reading (monitor reading)	dciReadVariables( )	Read data from specified inconsecutive addresses	O	O	O
10	Multipoint reading (monitor reading) (Adapted to command status)	dciReadVariablesS( )	Read data from specified inconsecutive addresses	X	X	O
11	Multipoint writing (monitor writing)	dciWriteVariables( )	Write data in specified inconsecutive addresses	O	O	O
12	Multipoint writing (monitor writing) (Adapted to command status)	dciWriteVariablesS( )	Write data in specified inconsecutive addresses	X	X	O
13	Batch start	dciBatchStart( )	Start all the CPUs existing in a configuration as a batch.	X	X	O
14	Batch initialize and start	dciBatchInitialStart( )	Initialize and start all the CPUs existing in a configuration as a batch.	X	X	O
15	Batch stop	dciBatchStop()	Stop all the CPUs existing in a configuration as a batch.	X	X	O
16	Batch reset	dciBatchReset()	Reset all the CPUs existing in a configuration as a batch.	X	X	O
17	ID code reading	dciReadIdCode()	Read CPU module information.	X	X	O
18	Start measurement of task execution time	dciStartMeasureTaskTime()	Start measurement of the task execution time.	X	X	O
19	Stop measurement of task execution time	dciStopMeasureTaskTime()	Stop measuremenr of the task execution time.	X	X	O
20	Read task execution time	dciReadTaskTime()	Read measurement results of the task execution time.	X	X	O
21	Clear task execution time	dciClearTaskTime()	Clear measurement results of the task execution time.	X	X	O

Section 5 Procedure for Making an ISV Application Program

<Programming flow chart>



Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-1 Initialize linkage function

<General>

This function initializes the environment of the data linkage function.

* For the function reset value of individual function, refer to “6-1-18 Function reset value”.

<Interface>

short far pascal dciInit (int nMode);

nMode: fixed to 1 (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

6-1-2 Exit linkage function

<General>

This function releases the environment of the data linkage function.

<Interface>

short far pascal dciExit (void);

<Function reset value>

= 0 (ended normally)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-3 Connect configuration

<General>

This function establishes the connection with a configuration.
When connection is completed successfully, this function returns the corresponding connection ID to pnConnectionId.
If connection ended in failure, this function returns “-1” as connection ID.

<Interface>

short far pascal dciConnectConfig (void *pConnectParam, int *pnConnectionId);
where
pConnectParam : Reference to connection parameter (IN)
pnConnectionId : Pointer to the area in which connection ID is stored (OUT)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

<Connection parameter (pConnectParam)>

The ID of connection I/F and that of the target to connect are set with **a single-byte space character** inserted in between them as delimiter.

Connecting method (device name)	ID of connection I/F	ID of target to connect
ISA PLC board 0 (FMV2SPL0)	ISAPLC0	None
ISA PLC board 1 (FMV2SPL1)	ISAPLC1	None
ISA PLC board 2 (FMV2SPL2)	ISAPLC2	None
ISA PLC board 3 (FMV2SPL3)	ISAPLC3	None
SX bus board 0 (FMV2SXB0)	SXBUS0	SX station No. of the target to connect
SX bus board 1 (FMV2SXB1)	SXBUS1	SX station No. of the target to connect
SX bus board 2 (FMV2SXB2)	SXBUS2	SX station No. of the target to connect
SX bus board 3 (FMV2SXB3)	SXBUS3	SX station No. of the target to connect
PCI PLC board 0 (FMV2SPP0)	PCIPLC0	None
PCI PLC board 1 (FMV2SPP1)	PCIPLC1	None
PCI PLC board 2 (FMV2SPP2)	PCIPLC2	None
PCI PLC board 3 (FMV2SPP3)	PCIPLC3	None

(Example of setting 1)
When ISA PLC board 0 is used:
Connection parameter = “ISAPLC0”

(Example of setting 2)
When connecting from SX bus board via CPU0 (SX station No. 254):
Connection parameter = “SXBUS0 254”

(Example of setting 3)
When PCI PLC board 0 is used:
Connection parameter = “PCIPLC0”

6-1-4 Disconnect configuration

<General>

This function disconnects configuration.

<Interface>

short far pascal dciDisconnectConfig (int nConnectionId);

nConnectionId : Connection ID obtained when connecting configuration (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-5 Data reading

<General>

This function reads data for specified number of words, starting with specified top address.
When data reading is completed, the number of words of read data and the data are returned.

* If a response command from the CPU is abnormal, the command status adapted function “dciReadDataS()” sets a status code of the response command for the reset information.

(Example) If an inexistent memory is accessed, 1092 or 1093 is set for the reset value.

Note: If an inexistent memory is accessed when the data reading function is used, or if data reading is executed while a project is being downloaded to the CPU, 0 (zero) is returned as the number of words of read data.

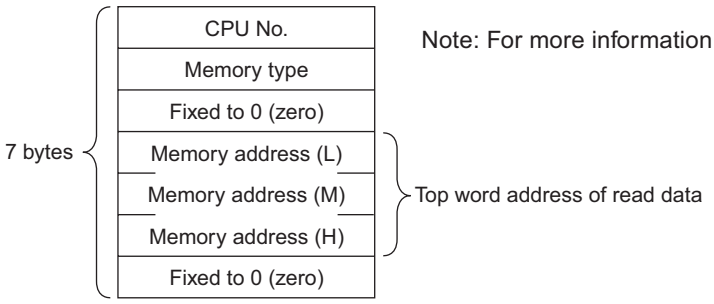
<Interface>

```
short far pascal dciReadData   ( int nConnectionId,
                                 void *pIcode,
                                 int *pnReadSize,
                                 void *pReadData );
<Command status adapted function>
short far pascal dciReadDataS ( int nConnectionId,
                                 void *pIcode,
                                 int *pnReadSize,
                                 void *pReadData );
```

where

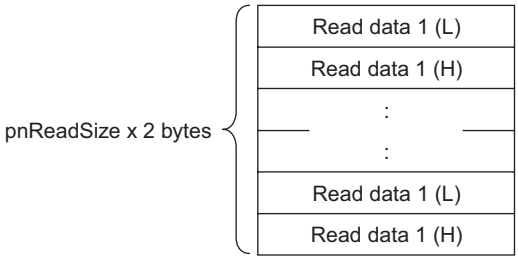
nConnectionId	: Connection ID obtained when connecting configuration (IN)
pIcode	: Pointer to the area in which ICODE is stored (IN)
pnReadSize	: Number of words of read data. Maximum 4096 words can be read (IN/OUT)
pReadData	: Pointer to the area in which read data is stored (OUT)

<Format of ICODE (pIcode)>



Note: For more information of memory type, refer to “7-1 Memory Type”.

<Format of read data (pReadData)>



<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-6 Data writing

<General>

This function writes data for specified number of words in the area starting with specified top address.
When data writing is completed, the number of words of written data and the data are returned.

* If a response command from the CPU is abnormal, the command status adapted function “dciWriteDataS()” sets a status code of the response command for the reset information.

(Example) If an inexistent memory is accessed, 1092 or 1093 is set for the reset value. If data writing is executed while a project is being downloaded to the CPU, 1059 is set for the reset value.

Note: If an inexistent memory is accessed when the data writing function is used, or if data writing is executed while a project is being downloaded to the CPU, 0 (zero) is returned as the number of words of written data.

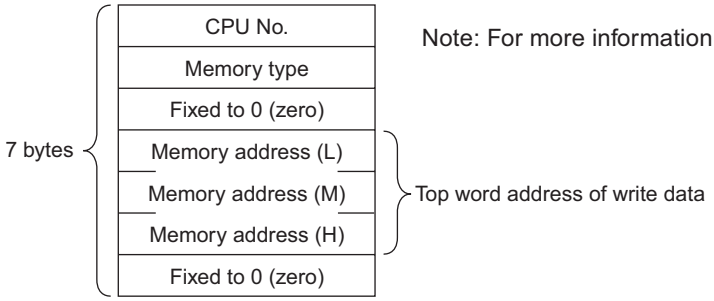
<Interface>

```
short far pascal dciWriteData   ( int nConnectionId,
                                  void *pIcode,
                                  int *pnWriteSize,
                                  void *pWriteData );
<Command status adapted function>
short far pascal dciWriteDataS ( int nConnectionId,
                                  void *pIcode,
                                  int *pnWriteSize,
                                  void *pWriteData );
```

where

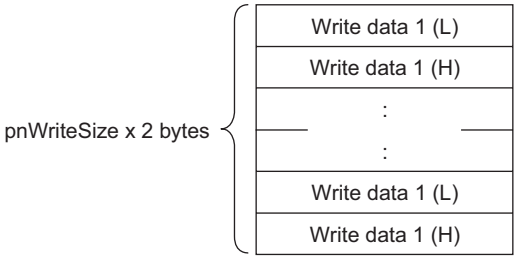
nConnectionId	: Connection ID obtained when connecting configuration (IN)
pIcode	: Pointer to the area in which ICODE is stored (IN)
pnWriteSize	: Number of words of written data. Maximum 4096 words can be written (IN/OUT)
pWriteData	: Pointer to the area in which written data is stored (IN)

<Format of ICODE (pIcode)>



Note: For more information of memory type, refer to “7-1 Memory Type”.

<Format of write data (pnWriteData)>



<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-7 Multipoint reading (monitor reading)

<General>

This function reads data from specified inconsecutive addresses.
When the reading of all the data is completed, the number of read points and the data are returned.

* If a response command from the CPU is abnormal, the command status adapted function “dciReadVariablesS()” sets a status code of the response command for the reset value.

Note: If an inexistent memory is accessed when the multipoint reading function is used, or if multipoint reading is executed while a project is being downloaded to the CPU, 0 (zero) is returned as the number of read points.

<Interface>

```
short far pascal dciReadVariables   ( int nConnectionId,
                                      void *pIcodeTbl,
                                      int *pnReadVarNum,
                                      void *pReadDataTbl );
```

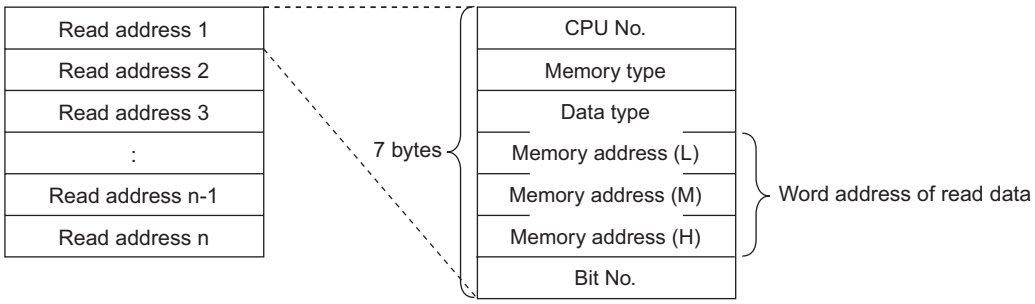
<Command status adapted function>

```
short far pascal dciReadVariablesS ( int nConnectionId,
                                      void *pIcodeTbl,
                                      int *pnReadVarNum,
                                      void *pReadDataTbl );
```

where

nConnectionId	: Connection ID obtained when connecting configuration (IN)
pIcodeTbl	: Pointer to the area in which ICODE is stored (IN)
pnReadVarNum	: Number of read data points. Maximum 2048 points of data can be read (IN/OUT)
pReadDataTbl	: Pointer to the area in which read data is stored (OUT)

<Format of ICODE (pIcodeTbl)>

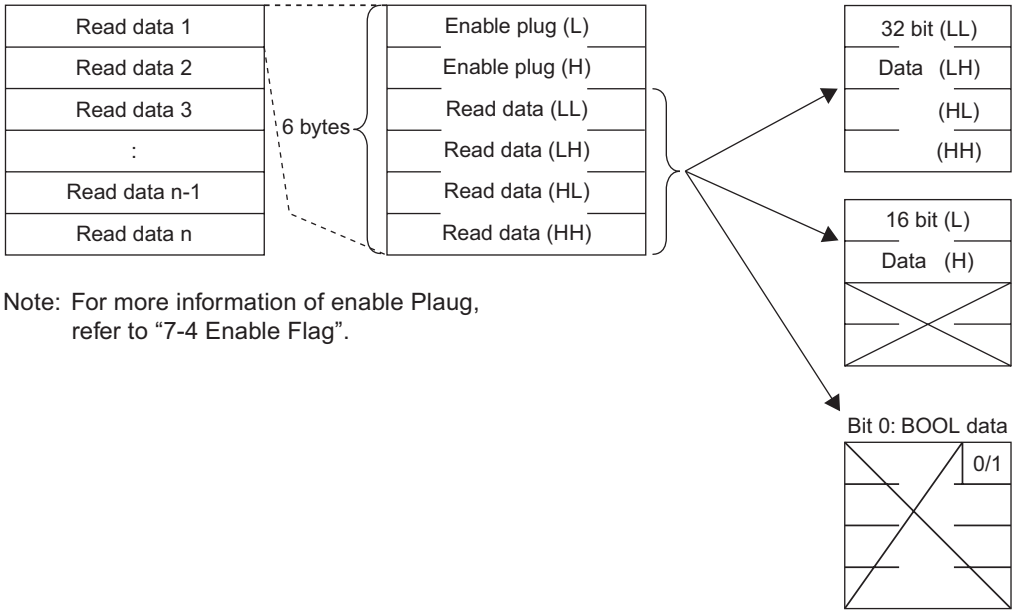


Note: For more information of type, refer to “Section 7 Description of ICODE”.

Section 6 Description of Individual Function

6-1 Description of Individual Function

<Format of read data (pReadDataTbl)>



Note: For more information of enable Plaug, refer to “7-4 Enable Flag”.

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-8 Multipoint writing (monitor writing)

<General>

This function writes data to specified inconsecutive addresses.
When the writing of all the data is completed, the number of write points and the data are returned.

* If a response command from the CPU is abnormal, the command status adapted function “dciWriteVariablesS()” sets a status code of the response command for the reset information.

(Example) If multipoint writing is executed while a project is being downloaded to the CPU, 1059 is set for the reset value.

Note: If an inexistent memory is accessed when the multipoint writing function is used, or if multipoint writing is executed while a project is being downloaded to the CPU, 0 (zero) is returned as the number of write points.

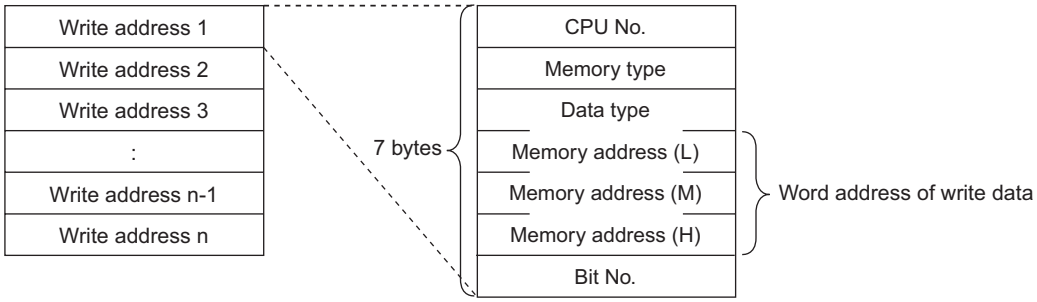
<Interface>

```
short far pascal dciWriteVariables   ( int nConnectionId,
                                       void *pIcodeTbl,
                                       int *pnWriteVarNum,
                                       void *pWriteDataTbl );
<Command status adapted function>
short far pascal dciWriteVariablesS ( int nConnectionId,
                                       void *pIcodeTbl,
                                       int *pnWriteVarNum,
                                       void *pWriteDataTbl );
```

where

nConnectionId	: Connection ID obtained when connecting configuration (IN)
pIcodeTbl	: Pointer to the area in which ICODE is stored (IN)
pnWriteVarNum	: Number of write data points. Maximum 2048 points of data can be written (IN/OUT)
pWriteDataTbl	: Pointer to the area in which write data is stored (OUT)

<Format of ICODE (pIcodeTbl)>

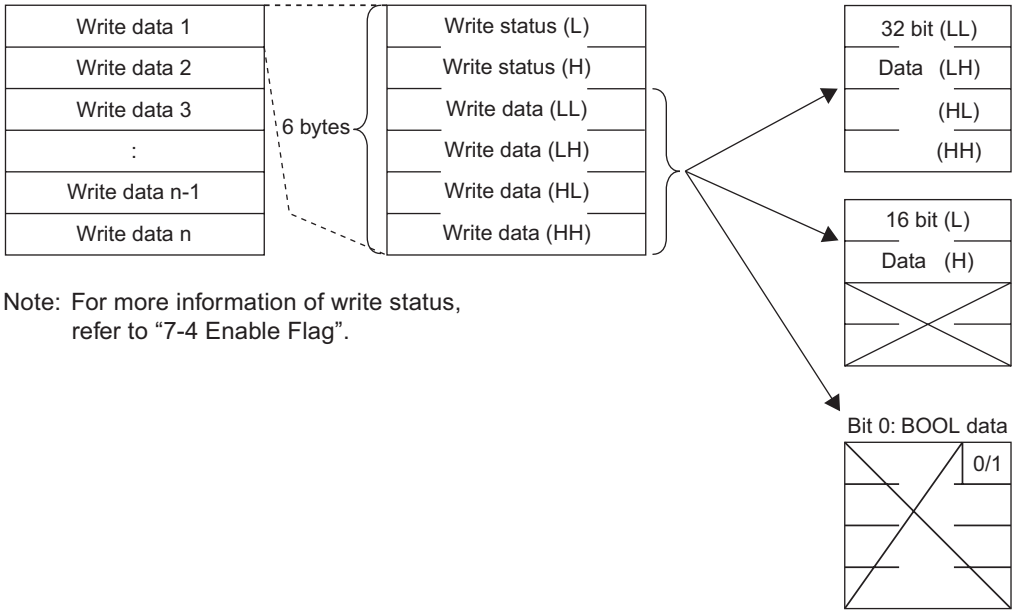


Note: For more information of type, refer to “Section 7 Description of ICODE”.

Section 6 Description of Individual Function

6-1 Description of Individual Function

<Format of write data (pWriteDataTbl)>



Note: For more information of write status, refer to “7-4 Enable Flag”.

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-9 Batch start

<General>

This function starts all the CPUs that exist in a configuration as a batch.
Individual CPU is cold-started or warm-started depending on its condition.

<Interface>

short far pascal dciBatchStart (int nConnectionId);

nConnectionId : Connection ID (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

6-1-10 Batch initialize and start

<General>

This function initializes and starts all the CPUs that exist in a configuration as a batch.
Individual CPU is cold-started or warm-started depending on its condition.

<Interface>

short far pascal dciBatchInitialStart (int nConnectionId);

nConnectionId : Connection ID (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

6-1-11 Batch stop

<General>

This function stops all the CPUs that exist in a configuration as a batch.

<Interface>

short far pascal dciBatchStop (int nConnectionId);

nConnectionId : Connection ID (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

6-1-12 Batch reset

<General>

This function resets all the CPUs that exist in a configuration as a batch.
Before executing reset, the CPUs inform whether or not request data is received normally.

<Interface>

short far pascal dciBatchReset (int nConnectionId);

nConnectionId : Connection ID (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-13 ID code reading

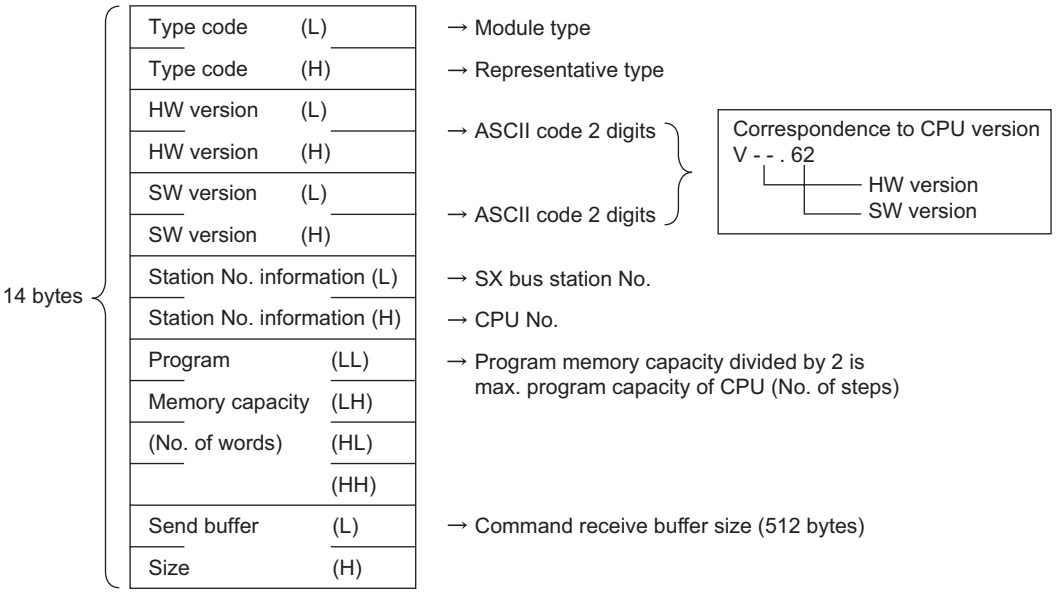
<General>

This function reads module information of the CPU module.
With version information included, you can check the version of the CPU.

<Interface>

```
short far pascal dciReadIdCode (  int nConnectionId,
                                  int nCpuNo,
                                  void *pModulInfo);
```

where
nConnectionId : Connection ID (IN)
nCpuNo : CPU No. (IN)
pModulInfo : Pointer to the area in which module information data is stored (OUT)



<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-14 Start measurement of task execution time

<General>

This function starts measurement of the task execution time. The measurement is continued until the “Stop measurement of task execution time” command is sent or the CPU is reset.

* If this command is sent while the CPU is stopped, measurement is started when the CPU is started again. If this command is sent during measurement, measurement data up to the point is cleared and then measurement is started.
* If the CPU is reset, measurement of the task execution time is stopped and measurement data is also cleared.

<Interface>

```
short far pascal dciStartMeasureTaskTime  ( int nConnectionId,
                                            int nCpuNo);
where
nConnectionId    : Connection ID (IN)
nCpuNo           : CPU No. (IN)
```

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

6-1-15 Stop measurement of task execution time

<General>

This function stops measurement of the task execution time. Even if this command ends, the measurement data is hold.

* The measurement data is cleared when the “Start measurement of task execution time” or “Clear task execution time” command is sent or the CPU is reset.
* If the CPU is reset, measurement of the task execution time is stopped and measurement data is also cleared.

<Interface>

```
short far pascal dciStopMeasureTaskTime  ( int nConnectionId,
                                           int nCpuNo);
where
nConnectionId    : Connection ID (IN)
nCpuNo           : CPU No. (IN)
```

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-16 Read task execution time

<General>

This function reads measurement results of the task execution time. Before this command is sent, be sure to send the "Start measurement of task execution time" command to start measurement of the task execution time .

* If the CPU is reset, measurement of the task execution time is stopped and measurement data is also cleared.

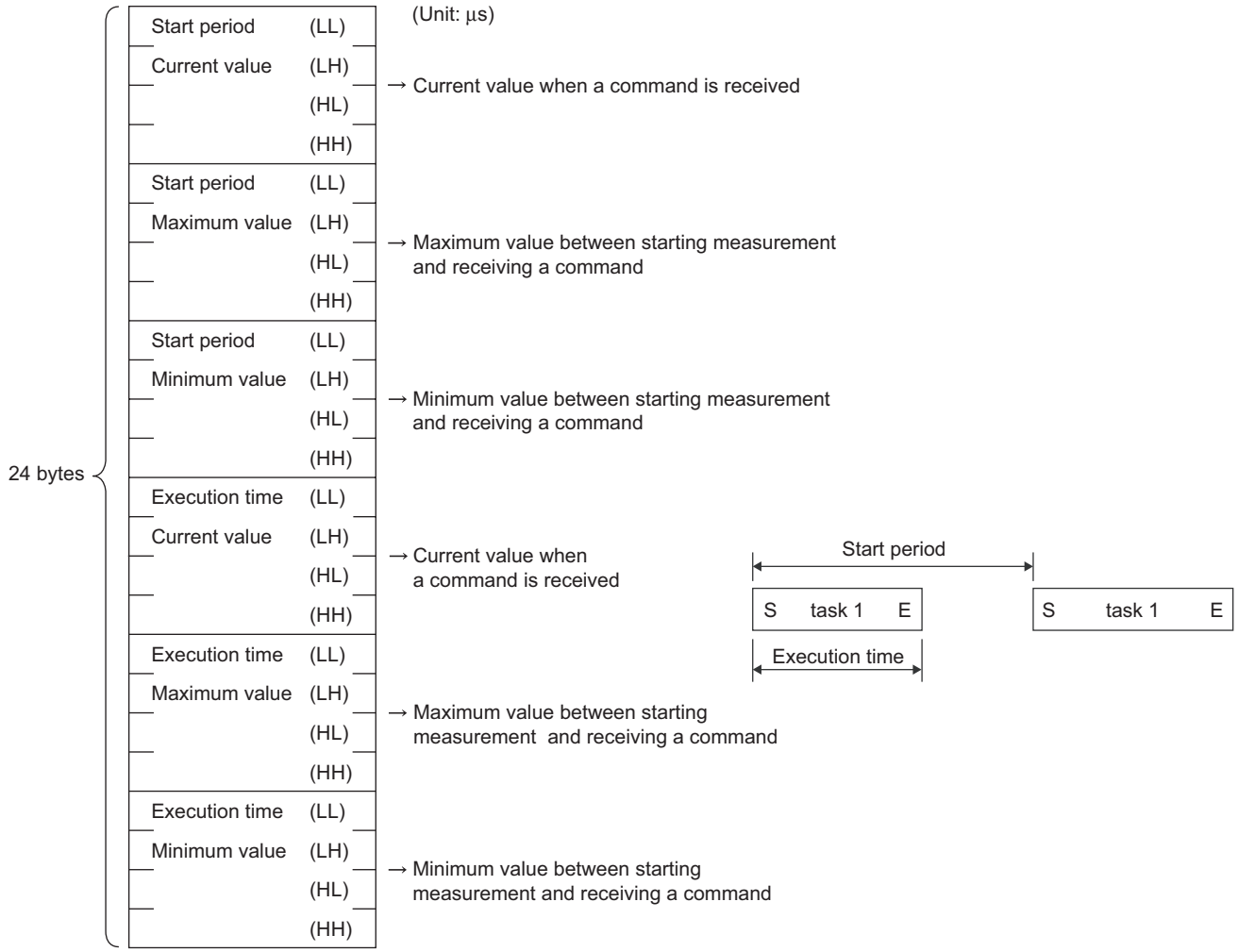
<Interface>

```
short far pascal dciReadTaskTime ( int nConnectionId,
                                   int nCpuNo
                                   int nTaskLevel,
                                   void *pTaskTime);
```

where

nConnectionId : Connection ID (IN)
nCpuNo : CPU No. (IN)
nTaskLevel : Task level (IN)
0000h → 0 level, 0001h → 1 level, 0002h → 2 level, 0003h → 3 level, 0004h → 4 level, 00FEh → Default level
pTaskTime : Pointer to the area in which task execution time data is stored (OUT)

<Format of task execution time data (pTaskTime)>



<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-17 Clear task execution time

<General>

This function clears measurement results of the task execution time (The measurement processing is continued). Before this command is sent, be sure to send the “Start measurement of task execution time” command to start measurement of the task execution time

* If the CPU is reset, measurement of the task execution time is stopped and measurement data is also cleared.

<Interface>

```
short far pascal dciClearTaskTime ( int nConnectionId,
                                    int nCpuNo);
```

where

nConnectionId	: Connection ID (IN)
nCpuNo	: CPU No. (IN)

<Function reset value>

= 0 (ended normally), > 0 (an error occurred that allows operation to continue), < 0 (an error occurred that prohibits operation to continue)

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-18 Function reset value

The following table shows the function reset values of individual function. Negative values mean the occurrence of an error that prohibits operation to continue.

Reset value	Description
0	Ended normally.
1	This function is not supported.
2	Insufficient memory capacity
3	Parameter error
4	The size of ICODE table is insufficient.
5	initLoaderCmd( ) is not yet processed.
102	Message manager is not yet started.
103	Failed to create a mail slot.
104	Connection to message manager is not established.
105	Cannot send data to message manager.
106	Operation status error
107	Communication time-out
108	ICODE data is wrong.
200	Initialization is already completed.
201	Initialization is not yet completed.
210	Illegal connection ID
220	Can no longer secure communication line.
221	Couldn't secure communication line.
222	Connection parameter error
223	Illegal value is set for the limit on loader command data size.
224	Loader command data size is insufficient.
225	Illegal value for loader command data size
256	Time-out occurred during internal socket receiving
-230	Message manager internal error
-231	Internal error
-250	Failed to create an internal socket.
-251	Failed to connect an internal socket.
-252	Failed to set up an internal socket.
-253	Failed to send data to internal socket
-254	Failed to receive data from internal socket.
-255	Internal socket was disconnected.
1024 - 1279 *	Loader command status error

* 1024 subtracted from a reset value is a loader command status code.

Section 6 Description of Individual Function

6-1 Description of Individual Function

6-1-19 Loader command status

For the function reset values between 1024 and 1279 on the preceding page, 1024 subtracted from each reset value is a loader command status. The following table shows details of the status codes.

Code	Status	Description
00h	Ended normally	The processing of command is completed successfully.
10h	CPU error	Command cannot be executed because an abnormality occurred on the CPU.
11h	CPU running	Command cannot be executed because the CPU is running.
12h	Command unexecutable	Command cannot be executed due to the key switch condition of the CPU.
20h	Undefined command	CPU received undefined command or mode.
22h	Parameter error	Setting error was found in command header part.
23h	Transmission interlocked	Transmission is interlocked by a command from other device.
28h	Processing a command	Requested command cannot be executed because other command is now being executed.
2Bh	Remote loader now processing	Requested command cannot be executed because the loader is now processing.
2Fh	Initialization not completed	Requested command cannot be executed because the system is now being initialized.
40h	Data setting error	Invalid data type or number was specified.
41h	Inexistent data	Specified data cannot be found.
44h	Memory address setting error	Specified address exceeds the valid range.
45h	Memory size over	Address + the number of read/write words exceed the valid range.
A0h	Command send destination setting error	No module exists at specified destination station number.
A2h	No response to command	No response data is returned from the remote module.
A4h	SX bus send error	Command cannot be communicated because an abnormality occurred on the SX bus.
A5h	SX bus send NAK	Command cannot be communicated because NAK occurred while sending data via SX bus.

Section 7 Description of ICODE

7-1 Memory Type

Type	Indent	Designated code
Input	%I□.○.△	00h
Output	%Q□.○.△	
Standard memory	%M□1.○.△	02h
Retain memory	%M□3.○.△	04h
System memory	%M□10.○.△	08h

□ : D (double-word) or W (word) or X (bit)
○ : Memory address
△ : Bit address

Note: For high-performance CPUs, the area of 2k words from the top of standard memory is a high-speed memory in which one access is processed in 20 ns. However, when said area is accessed from an external device, such as POD, one access requires 81 takts.

7-2 Data Type

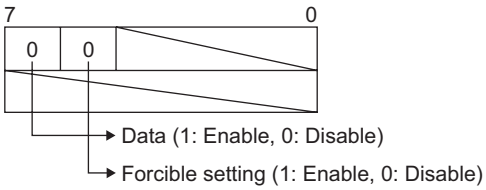
Name	Keyword	Bit width	Designated code
BOOL type	BOOL	1	04h
Integer type	INT	16	00h
Double precision integer type	DINT	32	08h
Unsigned integer type	UINT	16	02h
Unsigned double precision integer type	UDINT	32	09h
Real type	REAL	32	0Ah
Continuation time type	TIME	32	0Bh
16-bit string type	WORD	16	03h
32-bit string type	DWORD	32	0Fh

7-3 Bit No.

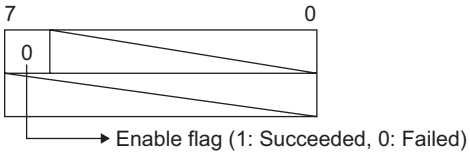
Name	Designated code	Name	Designated code
Bit 0	00h	Bit 8	08h
Bit 1	01h	Bit 9	09h
Bit 2	02h	Bit 10	0Ah
Bit 3	03h	Bit 11	0Bh
Bit 4	04h	Bit 12	0Ch
Bit 5	05h	Bit 13	0Dh
Bit 6	06h	Bit 14	0Eh
Bit 7	07h	Bit 15	0Fh

Section 7 Description of ICODE

7-4 Enable Flag



7-5 Write Status



7-6 Example of ICODE Setting

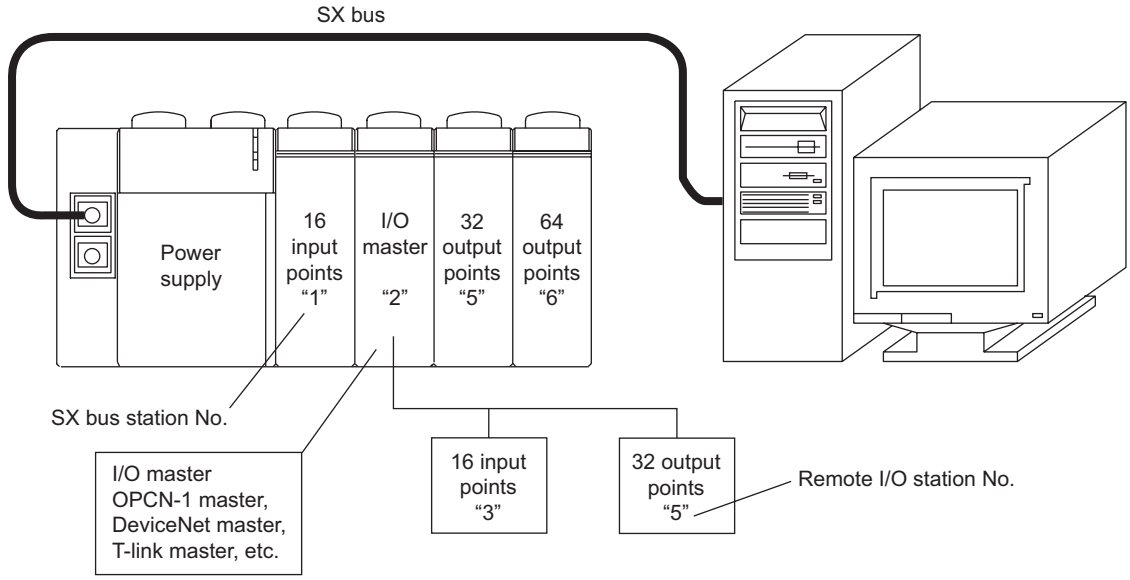
In the case of “Production result AT %MW3.100 :INT;”

Name	Set value
CPU No.	00h
Memory type	04h
Data type	00h
Memory address (L)	64h
Memory address (M)	00h
Memory address (H)	00h
Bit No.	00h

Section 7 Description of ICODE

7-7 Expression of I/O Area Addresses on ICODE

Input/output address is expressed by SX bus station No. and word No. (bit No.) per IEC. On the other hand, on ICODE it is expressed by the absolute address with respect to the top address of I/O memory area.



		Expression per IEC		Address on ICODE H	M	L
SX bus directly connected SX bus station No. = 1	SX station No. = 1 16 input points	%IW1.0	⇨	00	00	00
	For 32-bit boundary adjustment (See note 1)	Free	⇨	00	00	01
Remote I/O SX bus station No. = 2	Remote I/O station = 0 (free) (See note 2)	Free	⇨	00	00	02
	Remote I/O station = 1 (free) (See note 2)	Free	⇨	00	00	03
	Remote I/O station = 2 (free) (See note 2)	Free	⇨	00	00	04
	Remote I/O station = 3 16 input points	%IW2.3.0	⇨	00	00	05
	Remote I/O station = 4 (free) (See note 2)	Free	⇨	00	00	06
	Remote I/O station = 5 16 input points	%QW2.5.0	⇨	00	00	07
		%QW2.5.1	⇨	00	00	08
	For 32-bit boundary adjustment (See note 3)	Free	⇨	00	00	09
SX bus directly connected SX bus station No. = 5, 6	SX station No. = 5 32 output points	%QW5.0	⇨	00	00	0A
		%QW5.1	⇨	00	00	0B
	SX station No. = 6 64 output points	%QW6.0	⇨	00	00	0C
		%QW6.1	⇨	00	00	0D
		%QW6.2	⇨	00	00	0E
		%QW6.3	⇨	00	00	0F

Note 1: The I/O modules that are directly connected to SX bus are allocated in 32-bit boundary. Therefore, for a module with 16 input or output points, there occurs a free space of 16 bits. When there is a free space between an SX bus station No. and the station No. of the next module (In the above example, SX bus Nos. 3 and 4 are not used), the free SX bus station numbers do not occupy any input/output area.

Note 2: For remote I/O, each station occupies 16 bits, whether or not the station number actually exists.

Note 3: When the end address of a remote I/O station No. spreads beyond a 32-bit boundary, there occurs a free space of 16 bits.

Fuji Electric Co., Ltd.

Gate City Ohsaki, East Tower,
11-2, Osaki 1-chome, Shinagawa-ku, Tokyo 141-0032, Japan

E-mail: micrex-sx@fujielectric.co.jp
URL: http://www.fujielectric.com/

Issued as FE consolidated edition, June 2011