To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

USER'S MANUAL

RENESAS

78K/0 SERIES OS

MX78K0

FUNDAMENTAL

Document No. U12257EJ2V0UM00 (2nd edition)
(Previous No. EEU-1532)
Date Published May 1997 N
Printed in Japan

TRON is an abbreviation of The Realtime Operating system Nucleus.

ITRON is an abbreviation of Industrial TRON.

MS-DOS is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Other company names and/or product names are trademarks or registered trademarks of their respective companies.

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 800-366-9782
Fax: 800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

INTRODUCTION

Thank you for purchasing the 78K/0 Series Operating System (OS) MX78K0, the OS software for the NEC 78K/0 Series.

This manual explains the functions of the 78K/0 Series OS MX78K0.

[Target Users]

This manual assumes that the reader has read the target device's User's Manual at least once and has experience in software programming.

Note that this package includes the OS itself but does not include the CC78K0 Series C Compiler or the RA78K0 Series Assembler Package that are required when using this OS.

[Contents]

The contents of this manual are listed below.

[Legend]

The meanings of common symbols in this manual are described below.

…	:	Same format is repeated.
[]	:	Characters between brackets ([]) can be omitted.
⌈⌋	:	Characters between brackets (⌈⌋) can be omitted.
“ ”	:	Characters between double quotation marks are as shown.
‘ ‘	:	Characters between single quotation marks are as shown.
()	:	Characters between parentheses are as shown.
Boldface	:	Characters in boldface are as shown.
___	:	Underlining is used to indicate either important information or input character strings when describing use examples.
Δ	:	Indicates one or more blank spaces.
:	:	Indicates that program description is omitted
/	:	Divider
\	:	Back slash

[Related Documentation]

The following documents (User’s Manuals, etc.) are related to this manual.

Document	Document No.
RA78K Series Assembler Package, Language	EEU-1404
RA78K Series Assembler Package, Operation	EEU-809[Note]
RA78K Series Structured Assembler Preprocessor	EEU-1402
RA78K0 Assembler Package, Assembly Language, RA78K0 Ver. 3.10 or later	U11801E
RA78K0 Assembler Package, Operation, RA78K0 Ver. 3.10 or later	U11802E
RA78K0 Assembler Package, Structured Assembly Language, ST78K0 V3.10 or later	U11789E
CC78K0 C Compiler Language	U11518E
CC78K0 C Compiler Operation, CC78K0 Ver. 3.00 or later	U11517E
78K/0 Series Instruction Set	IEU-849[Note]
ideal-L Screen Editor, Introduction	U10094J[Note]

Note This document number is that of Japanese-version document.

[MEMO]

Phase-out/Discontinued

CONTENTS

LIST OF FIGURES

LIST OF TABLES

[MEMO]

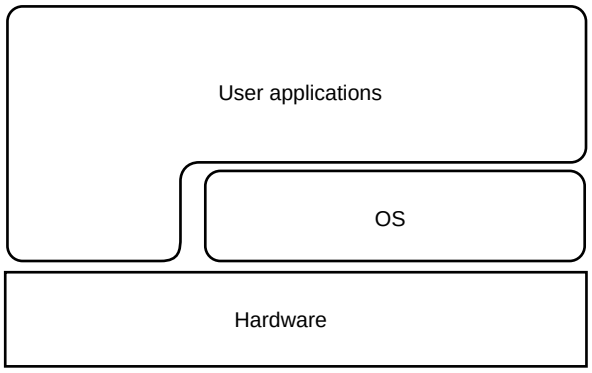
CHAPTER 1 GENERAL

This OS (Operating System) is the 78K/0 Series OS, which supports software development for applications that use limited memory resources in comparison to a real-time OS. This OS can be easily ported to the RX78K Series, the real-time OS for the 78K Series, since its specifications are a subset of the μITRON standard specifications.

1.1 Configuration

The system configuration is shown in Figure 1-1.

Figure 1-1. System Configuration



1.2 Features

The features of this OS for the 78K/0 Series are listed below.

1. **High-speed, compact design**
   Because the OS specifications are a μITRON subset, design is fast and compact.

2. **Elimination of task WAIT status**
   The task WAIT status has been eliminated to reduce the OS's memory usage. This curtails the OS's internal RAM usage.

3. **Elimination of preemption function**
   The preemption (forced interrupt of task) function has been eliminated to reduce the OS's memory usage. This curtails the OS's internal RAM usage. As a result, the task continues to operate until the ext_tsk() system call occurs.

4. **Slim-type OS and debug-type OS**
   This OS is available in two types: the "slim type" which does not detect errors after a system call occurs, and a "debug type" that detects such errors.
   Select the type that is most appropriate for the application being developed.

1.3 Overview of Functions

This OS provides the following functions.

◎ It provides various functions for system calls issued by application tasks.

◎ It provides management of the task execution sequence, including switching to the next task to be executed.

This OS includes the following management functions.

1. **Task management**
   It manages the start and termination of tasks, which are the units of processing under this OS.

2. **Event management**
   This includes event management to start tasks corresponding to events occurring inside or outside the system.

3. **Time management**
   This function manages a timer for starting tasks after a specified period of time.

4. **Interrupt management**
   It manages processes driven by interrupts as events.

1.4 Execution Environment

The following CPU is supported.

- 78K/0 Series

1.5 Application Development Environment

The system environment for developing applications is described below.

1. **Hardware**
   - PC-9801, 9821 Series (MS-DOS)
   - IBM PC Series (PC DOS) (preliminary)

2. **Software**
   - Assembler package
     NEC RA78K0 (for 78K/0)
   - Compiler package
     NEC CC78K0 (for 78K/0) Ver. 3.0 or later

The following software is required when using the provided idea-L file for development in the environment described above.

MS-Windows Ver. 3.1 or later
NEC idea-L Screen Editor Ver. 3.1 or later

CHAPTER 2 INSTALLATION

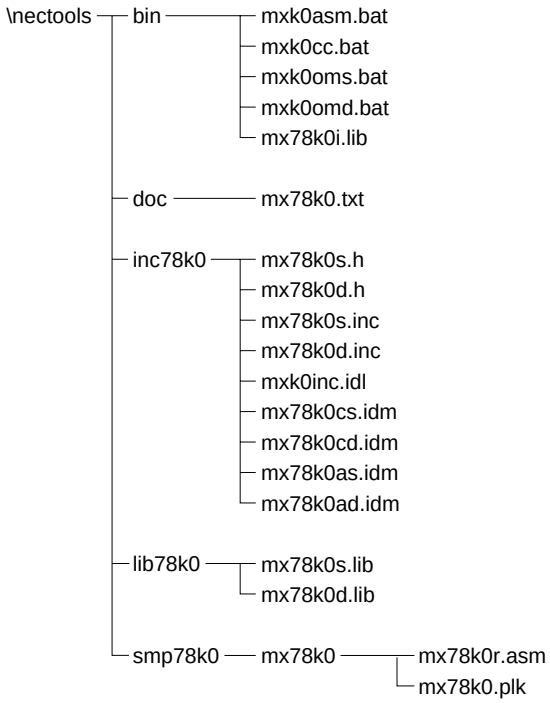
This chapter describes installation of this OS on a host machine.

2.1 Provided Files

Files are provided for this OS in two formats: object file format and source file format. Source files can be provided only when a large number of object files are purchased. The following directory configuration is provided for object file format and source file format.

2.1.1 Directory configuration for object file format

Figure 2-1. Directory Configuration for Object File Format



1. \nectools\bin directory

This directory includes various batch files for the make function and the ideal-L library file.

File name	mxk0asm.bat
Description	Batch file used to assemble user tasks and memory definition files.
Use Method	These batch files are used when assembling user tasks and memory definition files. The batch file contents can be changed to change the assembler's actions. When using these files, be sure to specify the chip type and file name (without the extension) as parameters. **(Example)** mxk0asmΔ014Δusr1 (target CPU is μPD78014, file name is usr1.asm)

File name mxk0cc.bat
Description Batch files used to compile user tasks
Use Method Use these batch files to compile user tasks. The batch file contents can be changed to change the compiler's options.
When using these files, be sure to specify the chip type and file name (without the extension) as parameters.
(Example) mxk0cc∆014∆usr1 (target CPU is μPD78014, file name is usr1.asm)

File name mxk0oms.bat
Description Batch files used to create slim type load modules
Use Method Link "mx78k0s.lib" with other required files (specified in "mx78k0.plk") to create a load module.
The contents of this file can be changed to change the linker options.
When using these files, be sure to specify the chip type as a parameter.
(Example) mxk0oms∆014∆usr1 (target CPU is μPD78014)

File name mxk0omd.bat
Description Batch files used to create debug type load modules
Use Method Link "mx78k0d.lib" with other required files (specified in "mx78k0.plk") to create a load module.
The contents of this file can be changed to change the linker options.
When using these files, be sure to specify the chip type as a parameter.
(Example) mxk0omd∆014∆usr1 (target CPU is μPD78014)

File name mx78k0i.lib
Description idea-L function library file
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

2. \nectools\doc directory

This directory includes files that contain caution notes and other information that is not in the User's Manual.

File name mx78k0.txt
Description These are text files that contain caution notes and other information not found in the User's Manual.

3. \nectools\inc78k0 directory

This directory contains assembly language and C-language include files.

File name mx78k0s.h, mx78k0cs.idm (for idea-L)
Description Include files (C language) for slim type
Use Method Include and use these files when coding source files in C, since they contain type declarations for system calls. When using these files, any content other than system call type declarations for the user system should be included in comments or deleted.

File name mx78k0d.h, mx78k0cd.idm (for idea-L)
Description Debug type include files (C language)
Use Method Include and use these files when coding source files in C, since they contain return code definitions and type declarations for system calls. When using these files, any content other than system call type declarations for the user system should be included in comments or deleted.

File name mx78k0s.inc, mx78k0as.idm (for idea-L)
Description Slim type include files (assembly language)
Use Method Include and use these files when writing assembly language source files, since they contain system call macros. When using these files, any unused system call macros should be converted to comments.

File name mx78k0d.inc, mx78k0ad.idm (for idea-L)
Description Debug type include files (assembly language)
Use Method Include and use these files when writing assembly language source files, since they contain return code definitions and system call macros. When using these files, any unused system call macros should be converted to comments.

File name mxk0inc.idl
Description Information file that idea-L uses to manage idea-L source files (.idm extension) in the same directory
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

4. \nectools\lib78k0 directory

This directory includes the nucleus' library files.

File name mx78k0s.lib
Description Slim type library file
Use Method Link this when creating a load module.

File name mx78k0d.lib
Description Debug type library file
Use Method Link this when creating a load module.

5. \nectools\smp78k0\mx78k0 directory

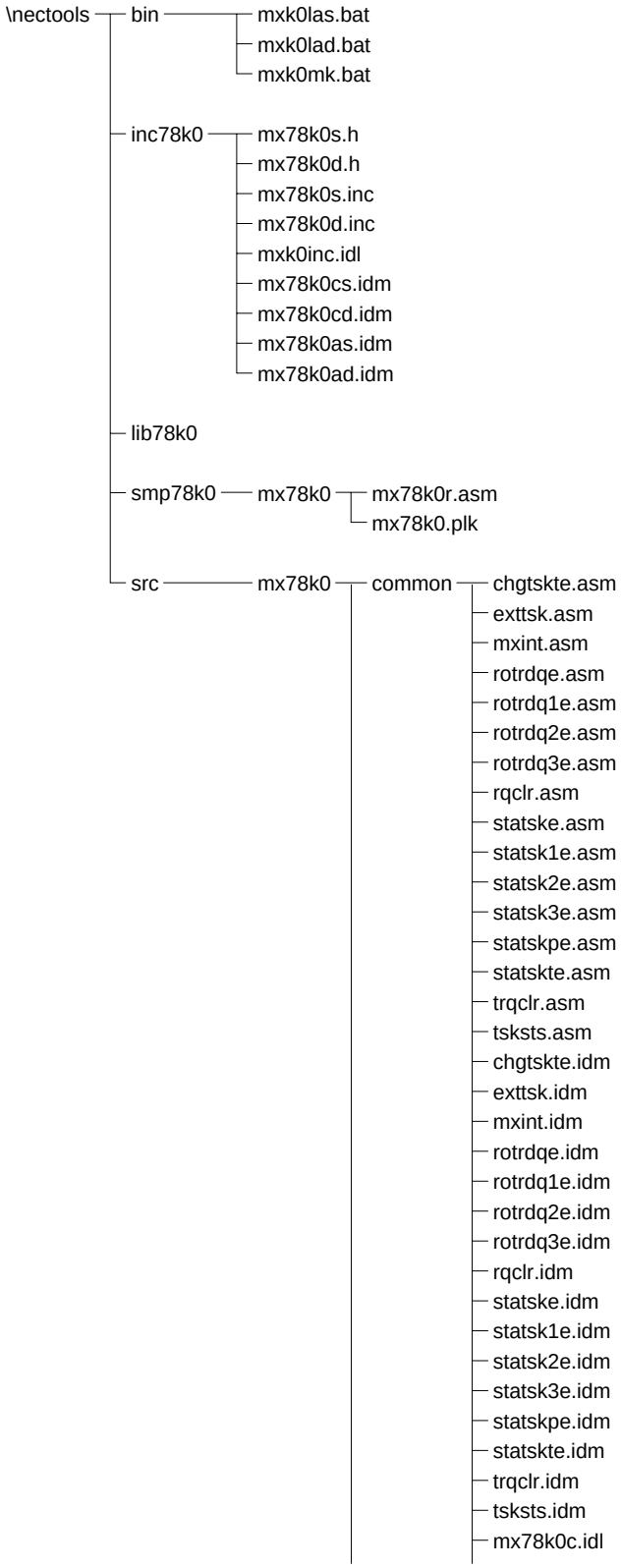
This directory includes sample files.

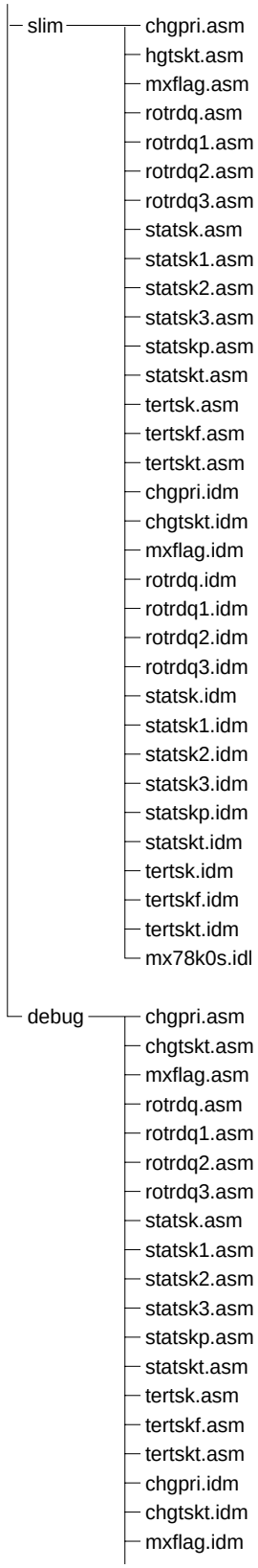
File name mx78k0r.asm
Description Sample file for memory definition files
Use Method This file is used to define the memory used by the OS. Change this sample file to suit the user system.

File name mx78k0.plk
Description Linker parameter file
Use Method This file contains the files to be linked when using "mxk0oms.bat" or "mxk0omd.bat" to create an object module. Use this file in addition to user files.

2.1.2 Directory configuration for source file format

Figure 2-2. Directory Configuration for Source File Format





- rotrdq.idm
- rotrdq1.idm
- rotrdq2.idm
- rotrdq3.idm
- statsk.idm
- statsk1.idm
- statsk2.idm
- statsk3.idm
- statskp.idm
- statskt.idm
- tertsk.idm
- tertskf.idm
- tertskt.idm
- mx78k0d.idl

1. \nectools\bin directory

This directory contains batch files for building a library.

File name	mxk0mk.bat
Description	Batch files for building a system call library
Use Method	These batch files can be used to build either debug type or slim type library files. Execute the batch files in the \nectools\src\mx78k0 directory and be sure to specify the chip type as a parameter. **(Example)** mxk0mkΔ014 (target CPU is μPD78014)

File name	mxk0as.bat, mxk0ad.bat
Description	Batch files used by “mxk0mk.bat” (not used by ordinary users)

2. \nectools\inc78k0 directory

This directory contains assembly language and C-language include files.

File name	mx78k0s.h, mx78k0cs.idm (for idea-L)
Description	Include files (C language) for slim type
Use Method	Include and use these files when coding source files in C, since they contain type declarations for system calls. When using these files, any unused system call type declarations for the user system should be converted to comments.

File name	mx78k0d.h, mx78k0cd.idm (for idea-L)
Description	Debug type include files (C language)
Use Method	Include and use these files when coding source files in C, since they contain return code definitions and type declarations for system calls. When using these files, any unused system call type declarations for the user system should be converted to comments.

File name mx78k0s.inc, mx78k0as.idm (for idea-L)
Description Slim type include files (assembly language)
Use Method Include and use these files when writing assembly language source files, since they contain system call macros. When using these files, any system call macros that are not used by the user system should be converted to comments.

File name mx78k0d.inc, mx78k0ad.idm (for idea-L)
Description Debug type include files (assembly language)
Use Method Include and use these files when writing assembly language source files, since they contain return code definitions and system call macros. When using these files, any system call macros that are not used by the user system should be converted to comments.

File name mxk0inc.idl
Description Information file that idea-L uses to manage idea-L source files (.idm extension) in the same directory
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

3. \nectools\lib78k0 directory

This is the directory for storing system call library fields that have been rebuilt by the user. There are no files in this directory.

4. \nectools\smp78k0\mx78k0 directory

This directory contains sample files.

File name mx78k0r.asm
Description Sample file for memory definition files
Use Method This file is for defining the memory used by the OS. Change this sample file to suit the user system.

File name mx78k0.plk
Description Linker parameter file
Use Method This file contains the files to be linked when using a batch file ("mxk0oms.bat" or "mxk0omd.bat") to create load modules that are provided in the object file format supply media. Use this file in addition to user files.

5. \nectools\src\mx78k0 directory

System call source files are entered according to type (slim or debug) into this directory.

\nectools\src\mx78k0\common directory

This directory contains source files that can be used for slim type or debug type.

File name exttsk.asm, exttsk.idm (for idea-L)
Description Source file for ext_tsk system call or this OS's task switching routine

File name mxint.asm, mxint.idm (for idea-L)
Description Source file for timer processing routine

File name rdqclr.asm, rdqclr.idm (for idea-L)
Description Source file for initialization routine that clears the ready queue

File name trqclr.asm, trqclr.idm (for idea-L)
Description Source file for initialization routine that clears the ready queue and timer queue

File name *.asm, *.idm (other than above) (for idea-L)
Description Source file for various system calls

File name mx78k0c.idl
Description Information file that idea-L uses to manage idea-L source files (.idm extension) in the common directory
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

6. \nectools\src\mx78k0\slim directory

Slim type source files are contained in this directory.

File name *.asm, *.idm (for idea-L)
Description Source file for various system calls

File name mx78k0s.idl
Description Information file that idea-L uses to manage idea-L source files (.idm extension) in the slim directory
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

7. \nectools\src\mx78k0\debug directory

Debug type source files are contained in this directory.

File name *.asm, *.idm (for idea-L)
Description Source file for various system calls

File name mx78k0d.idl
Description Information file that idea-L uses to manage idea-L source files (.idm extension) in the debug directory
Use Method For description of this file's use method, see "**idea-L SCREEN EDITOR, INTRODUCTION**".

2.2 Installation Steps

The method used to install this OS's supply medium into the host machine varies according to the user's development environment. The following description presents an example.

"A>" and "B>" in the input examples indicate the prompt and "↓" indicates return-key input.

1. **Supply medium set**
   Insert the supply medium (a floppy disk) into the floppy disk drive.

2. **Change current disk drive**
   Change the current disk drive to the drive in which the supply medium was inserted.

(Example) In this case, change to drive B.

```
A > B : ↓
```

3. **Transfer file group**
   Run the xcopy command to transfer the group of files that have been stored on the supply medium to the host machine.

(Example) In this case, transfer the files from drive B to A:\.

```
B > xcopy  /e  /v  b:\   a:\ ↓
```

4. **Verify file group**
   Run the dir command to verify that the file group has been transferred to the host machine. For details of these directories, see **"2.1 Provided Files"**.

(Example)

```
B > dir   a:\ ↓
```

5. **Set command search path**
   Add the directory (\nectools\bin) that contains the batch files for the make function to the command search path.

(Example) In this case, the method for setting a path variable in an environment definition file (such as autoexec.bat) is shown.

```
path = a:\nectools\bin ↓
```

2.3 Installation-related Cautions

1. If installing only the source file format

If you are installing only the source file format, document files (“mx78k0.txt”), batch files (“mxk0asm.bat”, “mxk0cc.bat”, “mxk0oms.bat”, “mxk0omd.bat”) associated with the object file format, and function library files (“mx78k0i.lib”) for idea-L are not installed. Therefore, before using any of these files, be sure to transfer them to the host machine from the supply medium that contains object file format files.

2. If a directory other than the root directory is specified as the transfer destination

If you have specified a directory other than the root directory as the transfer destination, the batch files for the make function may not operate normally. In this case, change the path specification in each corresponding batch file.

3. MS-DOS commands

For details of the MS-DOS commands that are used during installation, specification of the command search path, etc., see the “**MS-DOS USER’S MANUAL**”.

CHAPTER 3 TASK MANAGEMENT

Tasks are the elements from which application programs are configured for this OS. This OS processes these tasks in program units.

This chapter describes the task management methods and statuses used by this OS.

3.1 Overview of Tasks

Tasks are the smallest units for processing executed under the control of this OS. Startup, execution, and termination are all carried out in task units.

This OS processes tasks not according to task priority but rather according to their position in the ready queue. Up to 63 tasks can be chained to the ready queue.

Task switching is performed only after termination of the task currently being executed (only after the ext_tsk() system call has been issued).

3.2 Task Startup

The task that enters the RUN status first after system startup is the task for which the sta_tsk() system call is issued during the system initialization process. During system initialization, when tasks other than the first task to be started are started by the sta_tsk() system call, these tasks are sequentially chained to the ready queue after the first task to be started by the sta_tsk() system call. If one of these tasks is started by the sta_tskp() system call, it is chained to the start of the ready queue, which means it enters the RUN status before the first task started by the sta_tsk() system call.

Tasks that are not started by the sta_tsk() system call during the system initialization process are started and chained to the ready queue when the sta_tsk() system call or other system calls that start tasks are issued during task execution. (See **"CHAPTER 10 SYSTEM INITIALIZATION PROCESS"** for details of system initialization.)

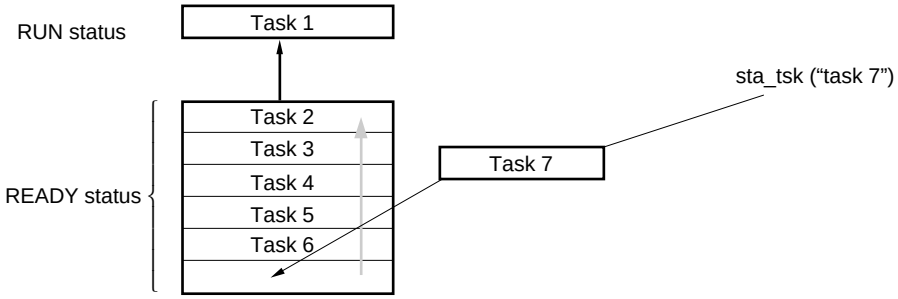
In addition to the sta_tsk() system call, the other task-starting system calls included in this OS are the sta_tskp() system call, which chains the specified task to the ready queue so that it is executed after the task currently being executed; the sta_tskt() system call, which starts a specified task after a specified amount of time; and the sta_tskf() system call, which starts a specified task according to the bit pattern of a specified event flag.

These system calls are described below.

3.2.1 Start by “sta_tsk()”

The sta_tsk() system call executes a process that chains the specified task to the end of the ready queue. The specified task then remains (in READY status) in the ready queue until its turn to be executed.

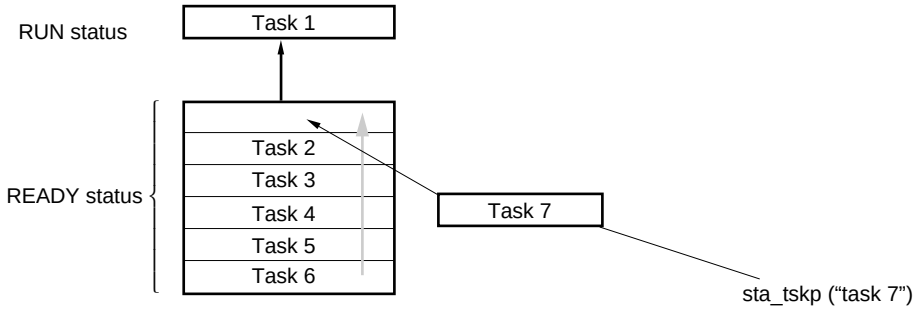
Figure 3-1. Description of “sta_tsk()”



3.2.2 Start by “sta_tskp()”

The sta_tskp() system call chains the specified task to the ready queue so that it is executed after the task that is currently being executed. Execution of this task begins after termination of the task that is currently being executed.

Figure 3-2. Description of “sta_tskp()”



3.2.3 Start by “sta_tskt()”

The sta_tskt() system call sets a specified task to be executed after a specified amount of time. The specified task is first chained to the timer queue. Then, after the specified amount of time has elapsed, the specified task is chained to the start of the ready queue and deleted from the timer queue. This task is the next task to be executed after the task that is being executed at the time when this task is chained to the ready queue. Up to 62 tasks can be chained to the timer queue.

Figure 3-3. Description of “sta_tskt()” (1/2)

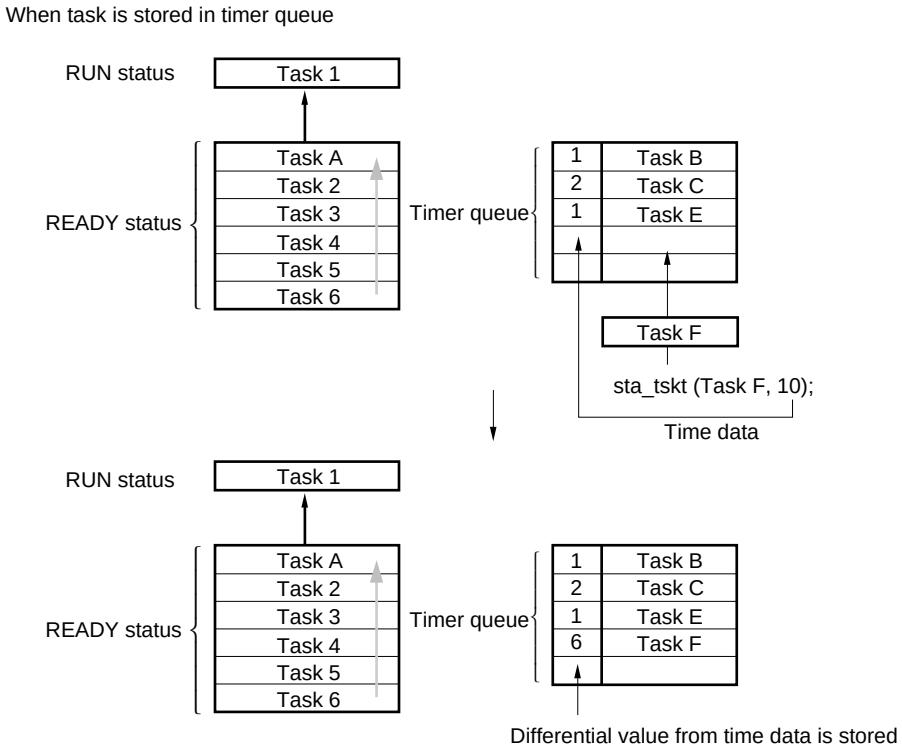
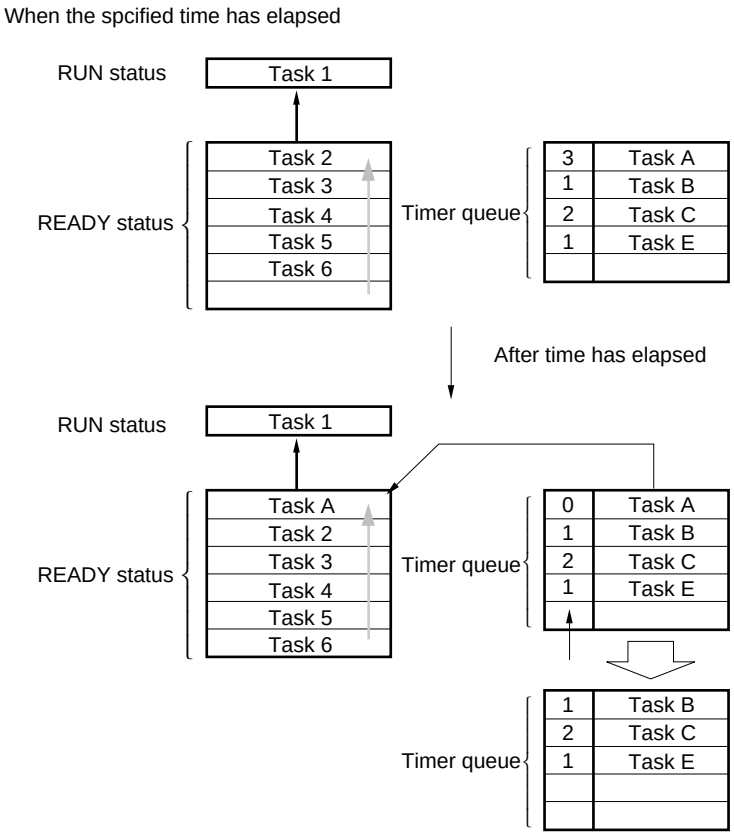


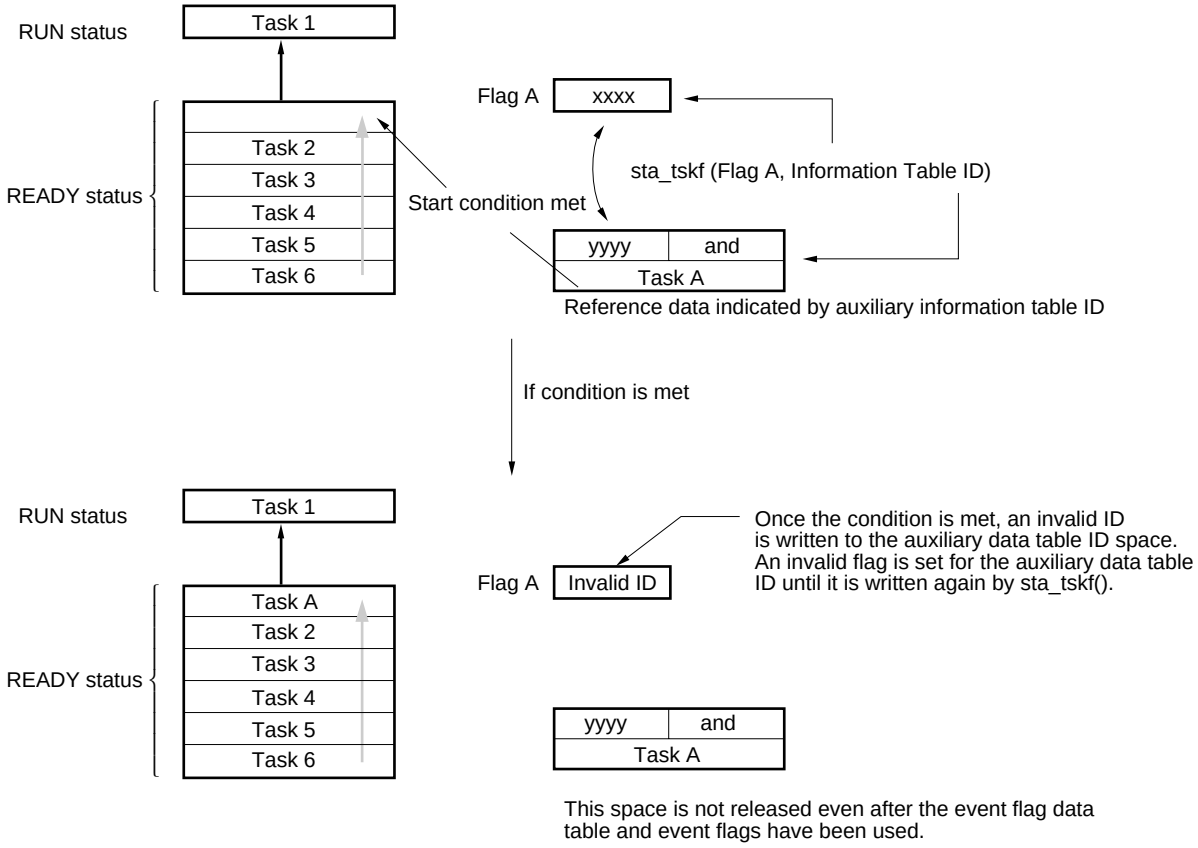
Figure 3-4. Description of “sta_tskt()” (2/2)



3.2.4 Start by “sta_tskf()”

The sta_tskf() system call registers startup tasks to specified event flags. A task registered to an event flag is chained to the start of the ready queue when the specified bit pattern that is issued by the set_flg() or rpl_flg() system call matches the wait bit pattern.

Figure 3-5. Description of “sta_tskf()”



3.2.5 Multiple Chaining of Tasks to Ready Queue

The same task can be chained repeatedly to the ready queue. Note, however, that the OS does not keep track of how many have been chained.

If you would rather not chain the same task several times to the ready queue, see the following. (For details of the ter_tsk() and tsk_sts() system calls, see **"CHAPTER 7 SYSTEM CALLS"**.)

(Example) When it is possible to chain the specified task several times

```
void task1()
{
    unsigned char status;
        :
    Processing of task1;
        :
    tsk_sts(&status,task2)          /* Check status of task2 */
    while(status == TTS_RDY)
    {
        ter_tsk(task2);             /* Issue ter_tsk for task2 */
        tsk_sts(&status,task2);     /* Check status of task2 */
    }
    sta_tsk(task2);                 /* Chain task2 to ready queue */
        :
    Processing of task1;
        :
    ext_tsk();
}
```

Just before the specified task is chained to the ready queue, use the tsk_sts system call to check whether or not the specified task has already been chained to the ready queue.

If the specified task has already been chained to the ready queue, the ter_tsk system call aborts the new chaining operation.

This method enables the specified task to be chained to the ready queue only after the same task has been completely removed from the ready queue.

3.3 Task Statuses

The statuses of the tasks that operate under this OS change during the course of their operation. This section describes the statuses taken by the tasks and how the statuses are managed.

Under this OS, tasks can take the following three statuses.

◇ **RUN status**

While in this status, the task is given the CPU access right by the OS and the corresponding program is executed. Only one task in the system can be in the RUN status at a time.

◇ **READY status**

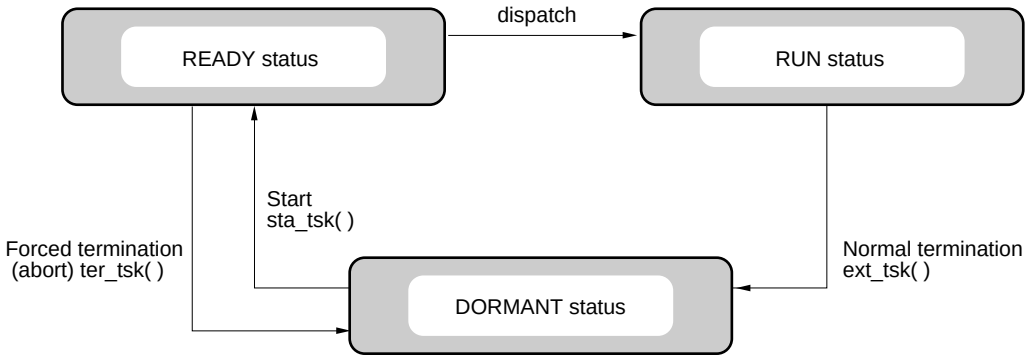
The task is chained to the READY queue, awaiting its turn for execution. When a task in the RUN status has been completed, the task at the start of the ready queue enters the RUN status.

◇ **DORMANT status**

All tasks that are neither in the RUN status nor in the READY status are in the DORMANT status.

Tasks that have been chained to the timer queue and tasks for which event flag-triggered startup has been specified are both in the DORMANT status.

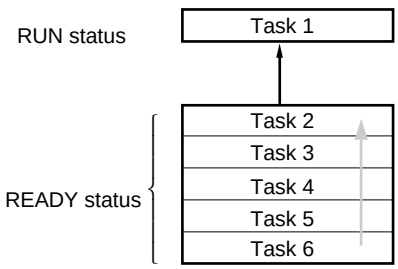
Figure 3-6. Task Status Changes



- Task management for RUN status

  Under this OS, the task that is chained to the start of the ready queue is the next task to enter the RUN status. Once it enters the RUN status, the task is removed from the ready queue.

Figure 3-7. Tasks in RUN and READY Statuses



- Management of tasks in READY status
  Under this OS, tasks that are changed to READY status by a task-starting system call such as “sta_tsk()” are managed after being chained to the READY queue. The task at the start of the ready queue will be executed after the task that is currently being executed.

- Management of tasks in DORMANT status
  Tasks that are in DORMANT status are not managed under this OS. However, the OS does manage tasks that have been chained to the timer queue and tasks for which event flag-triggered startup has been specified.

3.4 Task Dispatching

“Dispatching” refers to switching a task from the start of the ready queue to the RUN status. The mechanism for this process is called the dispatcher. The dispatcher starts when the task being executed has been completed.

3.5 Task Termination

Task termination refers to switching a task to the DORMANT status. The following system calls are used to terminate tasks.

- System calls that include a function for terminating the current task.

  ext_tsk(), sta_tske(), sta_tsk1e(), sta_tsk2e(), sta_tsk3e(), sta_tskpe(), sta_tskte(), chg_tskte(), rot_rdqe(), rot_rdq1e(), rot_rdq2e(), rot_rdq3e()

- System calls that include a function for terminating specified tasks.

  ter_tsk(), ter_tskt(), ter_tskf()

For description of the each system call’s functions, see **“CHAPTER 7 SYSTEM CALLS”**.

CHAPTER 4 EVENT FLAGS

Under this OS, event flags are used to start tasks corresponding to an event that occurs within or outside of the system.

Although several event flags can be set in the system, event flag-triggered startup can be set for only one task at a time. Several event flags can be set for the same task if they are all placed consecutively in RAM. This group of consecutive event flags are called the “event flag table”. In C language, a structure is used to define this table. In assembly language, the table is created by allocating a group of consecutive event flags.

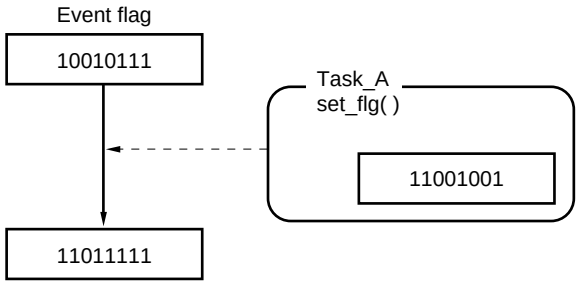
In addition, event flags require auxiliary data, such as definitions of tasks to be started, wait bit patterns, and wait conditions. Like the event flags themselves, several sets of event flag auxiliary data can be set consecutively in RAM. Unlike event flags, event flag auxiliary data can also be set consecutively in ROM. The group of consecutively set registered event flag auxiliary data is called the “event flag data table.” ID numbers are ordinal numbers (1, 2, 3 …) that are allocated to event flag auxiliary data in this table according to the order in which they were registered. The maximum ID value is 0xfe.

4.1 Setting Event Flags

Under this OS, event flags are set by the set_flg() system call.

When issuing the set_flg() system call, the bit pattern to be set is specified as a parameter. However, the bit pattern that is set for the event flag has a value that is the logical sum (OR) of the bit pattern that existed before the system call was issued and the bit pattern specified by the system call.

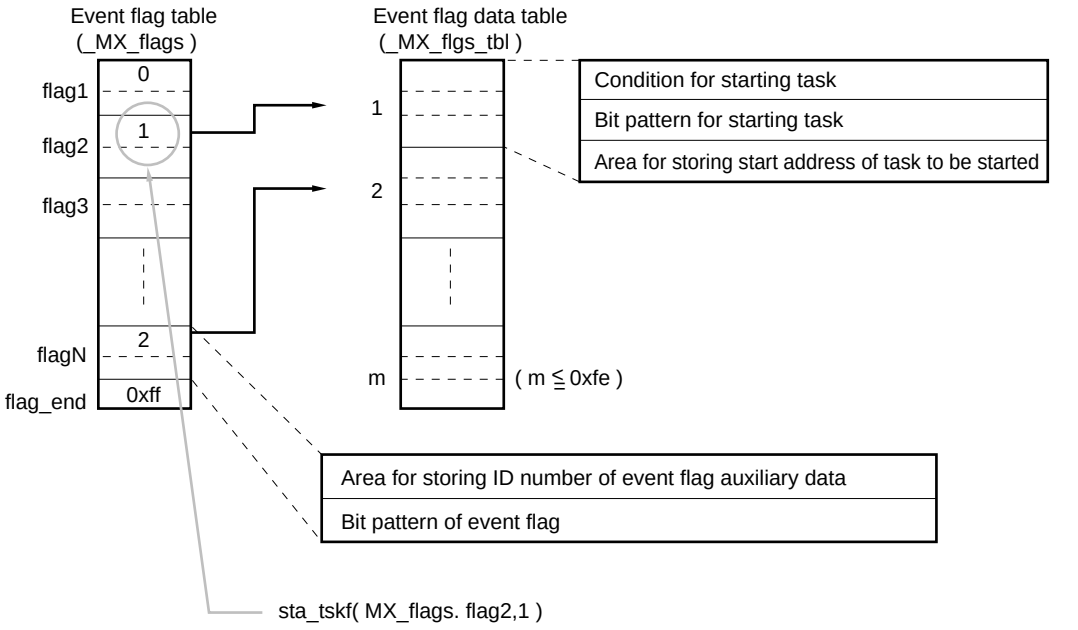
Figure 4-1. Setting of Event Flag



4.2 Registration of Task Corresponding to an Event Occurrence

Event flag auxiliary data ID numbers must be specified by issuing the sta_tsk() system call to enable registration of a task to corresponding to an event occurrence. Since the event flag data table is not created by a system call, the user must directly set the data to the event flag data table. For details, see **"10.3.6 Allocation of event flag"**.

Figure 4-2. Task Registration



There are two types of task start conditions.

- OR condition The specified task is started if at least one of the flags having the specified bit pattern of "1" is set.
- AND condition The specified task is started if all of the flags having the specified bit pattern are set.

4.3 Starting of Task Corresponding to Event Occurrence

The status of a task registered for an event flag changes from DORMANT to READY when the event flag's bit pattern is set by either the set_flg() or rpl_flg() system call and when the task's start condition has been met.

At that time, the task that has been registered for the event flag is chained to the start of the ready queue. An invalid ID is written to the area where event flag auxiliary data ID numbers are stored and the event flag's bit pattern is cleared to all zeros. Therefore, even if the event flag's bit pattern is set again, the task corresponding to that event flag will not be started when a new sta_tskf() system call is issued unless new auxiliary data has been registered for that event flag.

Caution After registering a task for an event flag, another task can be registered for the same event flag, but only the last task to be registered is valid.

CHAPTER 5 TIME MANAGEMENT

This chapter describes the timer used by this OS.

This OS uses timer interrupts to perform delayed task startup based on the timer's time value.

5.1 General

This chapter describe this OS's time management function and the timer interrupts upon which it is based.

This OS includes a function for delayed starting of tasks (chaining a task to the ready queue after a specified amount of time has elapsed). An 8/16-bit interval timer is used to implement this function.

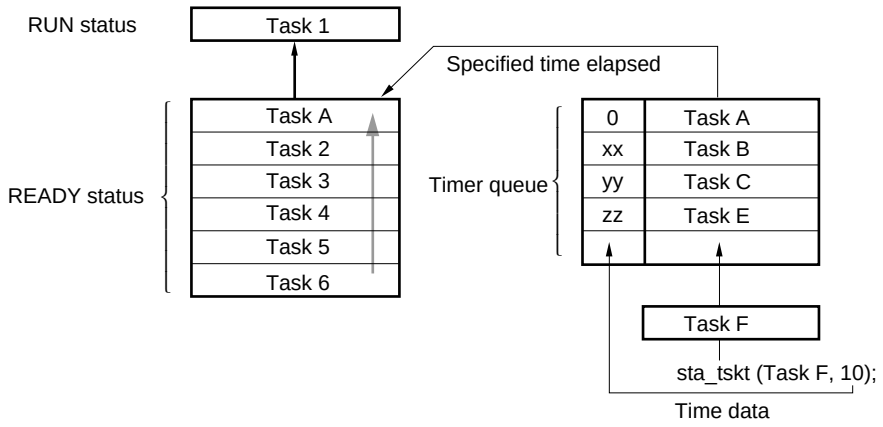
To initialize the interval timer (to set the interval value or select the interval timer), you must use the corresponding part of the system reset routine.

For details of interval timer initialization, see the User's Manual for the CPU being used.

5.2 Delayed Startup

This OS includes the sta_tskt() system call that starts the specified task after a specified amount of time has elapsed. When the sta_tskt() system call is issued, the specified task is first chained to the timer queue. Then, after the specified amount of time has elapsed, it is chained to the start of the ready queue (for details of the sta_tsk() system call, see **"7.3.12 sta_tsk"**).

Figure 5-1. Description of "sta_tskt()"



5.3 Revision of Startup Time

This OS includes the chg_tskt() system call that is used to change startup times.

First, the chg_tskt() system call searches to determine whether or not the specified task already exists in the task timer queue or ready queue.

If the specified task already exists, it is removed from the queue and is again chained to the timer queue with a newly specified time value as the startup time.

5.4 Multiple Chaining of Tasks to Timer Queue

The same tasks can be chained repeatedly to the timer queue. Note, however, that the OS does not keep track of how many have been chained.

To avoid chaining the same task several times to the timer queue, use the chg_tskt() system call instead of using the sta_tskt() system call to chain the task to the timer queue. (For details of the chg_tskt() system call, see **“CHAPTER 7 SYSTEM CALLS”**.)

CHAPTER 6 INTERRUPT MANAGEMENT

Interrupt management is the management of processes driven by interrupts as events. The routine for such interrupt-driven servicing is called the interrupt handler. This handling of interrupts is done independently of task management.

This OS does not initialize interrupt requests, so they must be initialized by the user. The user is also responsible for entering resets from the interrupt handler.

6.1 Position of Interrupt Handler

The interrupt handler is a interrupt servicing program that is run when an interrupt occurs. To ensure its responsiveness to interrupts, this servicing must be completed as promptly as possible. Therefore, the interrupt handler has execution priority over any task and even over the OS.

6.2 Processing within the Interrupt Handler

The interrupt handler's processing is performed directly when an interrupt occurs rather than via the OS. Accordingly, the following limitations and procedures apply to this processing.

1. **Save register**
   The user must use the interrupt handler to save or reset registers. This is because the interrupt handler processes directly without using the OS, and therefore must be started directly. Also, the user should check that the interrupt handler's processing does not use any register banks that are being used by the OS.

2. **Restriction on issuing system calls**
   The interrupt handler's processing must be high-speed processing. This means that system calls issued by the interrupt handler must be completed quickly.
   Issuing system calls may become difficult, such as when the interrupt handler attempts to issue a system call for an interrupt while another system call is being processed. Therefore, a restriction is placed on the types of system calls that can be issued by the interrupt handler.
   The following system calls can be issued by the interrupt handler.

   sta_tskp(), sta_tskf(), set_flg(), clr_flg(), rpl_flg()

   Normal operation is not guaranteed when a system call other than one of the above is issued by the interrupt handler. For description of each system call's functions, see **"CHAPTER 7 SYSTEM CALLS"**.)

3. **Termination of interrupt handler**
   The interrupt handler is terminated by the "RETI" instruction, which is a CPU instruction. It may be necessary to reset the register before issuing the "RETI" instruction.

6.3 Interrupts Used by the OS

Basically, this OS does not use interrupts. However, certain timer interrupts are used by the OS's time management function. The user can freely set timer interrupts that are used for time management.

To prevent misoperation of the time management function due to multiple interrupts, use the timer interrupts that are used exclusively for calling the OS's time management routines.

[MEMO]

CHAPTER 7 SYSTEM CALLS

System calls are OS services or call procedures that are provided so that tasks can execute various operations. System calls are used for operation such as starting or terminating tasks and setting or resetting event flags.

This section briefly describes the system call functions and how they are entered.

7.1 Overview of Functions

System calls are divided into the following two groups according to their functions.

- **Task management system calls**

  Task management system calls are the group of system calls that execute operations such as starting or terminating tasks. The system calls in this group are listed below.

  sta_tsk : Starts the specified task
  sta_tske : Starts the specified task and terminates the current task
  sta_tskp : Starts the specified task as the task following the current task
  sta_tskpe : Starts the specified task, then terminates the current task. The specified task is the task that is executed following the current task.

  ter_tsk : Forcibly terminates (aborts) a specified task in the ready queue

  sta_tskt : Starts the specified task after the specified time has elapsed
  sta_tskte : Sets the specified task to be started after the specified time has elapsed, then terminates the current task

  chg_tskt : Changes the startup time for the specified task
  chg_tskte : Changes the startup time for the specified task, then terminates the current task

  ter_tskt : Forcibly terminates a specified task in the timer queue or ready queue
  sta_tskf : Registers a task specified to be started by an event flag to an event flag

  ter_tskf : Forcibly terminates a task specified to be started by an event flag or a specified task in the ready queue

  rot_rdq : Rotates the current task to the end of the ready queue

  rot_rdqe : Rotates the current task to the end of the ready queue, then terminates the current task

  tsk_sts : Gets the status of the specified task

  chg_pri : Executes the specified task as the task following the current task

  ext_tsk : Terminates the current task

In addition, the following system calls are provided for sta_tsk(), sta_tske(), rot_rdq(), or rot_rdqe() to speed up the search operations by ter_tsk() and tsk_sts().

sta_tsk1 : Concurrent with sta_tsk() operations, this system call sets up a high-speed search using ter_tsk.
sta_tsk2 : Concurrent with sta_tsk() operations, this system call sets up a high-speed search using tsk_sts.
sta_tsk3 : Concurrent with sta_tsk() operations, this system call sets up a high-speed search using ter_tsk and tsk_sts.

sta_tsk1e : Concurrent with sta_tske() operations, this system call sets up a high-speed search using ter_tsk.
sta_tsk2e : Concurrent with sta_tske() operations, this system call sets up a high-speed search using tsk_sts.
sta_tsk3e : Concurrent with sta_tske() operations, this system call sets up a high-speed search using ter_tsk and tsk_sts.

rot_rdq1 : Concurrent with rot_rdq() operations, this system call sets up a high-speed search using ter_tsk.
rot_rdq2 : Concurrent with rot_rdq() operations, this system call sets up a high-speed search using tsk_sts.
rot_rdq3 : Concurrent with rot_rdq() operations, this system call sets up a high-speed search using ter_tsk and tsk_sts.

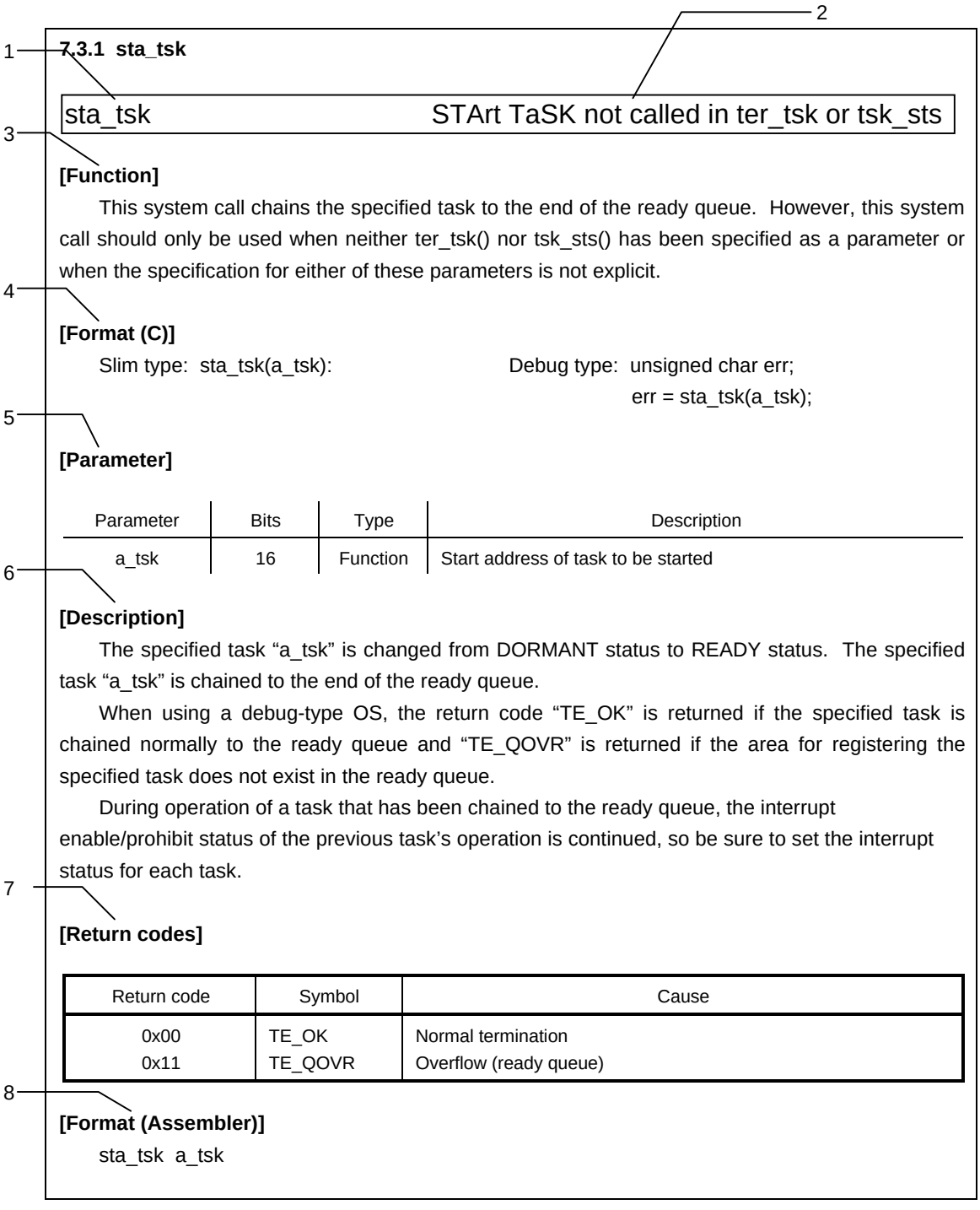
rot_rdq1e : Concurrent with rot_rdqe() operations, this system call sets up a high-speed search using ter_tsk.
rot_rdq2e : Concurrent with rot_rdqe() operations, this system call sets up a high-speed search using tsk_sts.
rot_rdq3e : Concurrent with rot_rdqe() operations, this system call sets up a high-speed search using ter_tsk and tsk_sts.

- **Event flag-related system calls**
  Event flag-related system calls comprise a group of system calls that manipulate flags to start tasks in response to events that occur as internal or external to the system. The system calls that belong to this group are described below.

set_flg : Sets any bit in an event flag.
clr_flg : Clears any bit in an event flag.
rpl_flg : Replaces any bit pattern in an event flag.

7.2 System Call Specifications

This section describes the details of system call specifications, including system call parameters and functions. These specifications are entered using the format shown below.



7.3.1 sta_tsk

sta_tsk STArt TaSK not called in ter_tsk or tsk_sts

[Function]

This system call chains the specified task to the end of the ready queue. However, this system call should only be used when neither ter_tsk() nor tsk_sts() has been specified as a parameter or when the specification for either of these parameters is not explicit.

[Format (C)]

Slim type: sta_tsk(a_tsk): Debug type: unsigned char err;
err = sta_tsk(a_tsk);

[Parameter]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

The specified task “a_tsk” is changed from DORMANT status to READY status. The specified task “a_tsk” is chained to the end of the ready queue.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and “TE_QOVR” is returned if the area for registering the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tsk a_tsk

1. Name of system call
2. Formal name of system call
3. Function. Overview of system call’s function

4. Format (C). Format for issuing system call from C language. If the format differs for slim type and debug type, each must be entered separately.
5. Parameters. This refers to the types and sizes (in bits) of parameters used when issuing system calls. The meaning and range of effective values for each parameter is also described. “None” is entered when no parameters are required.
6. Description of system call. This describes the system call’s processing and the return codes that are used for debug type OSs.
7. Return codes. The return code values (in hexadecimal code), symbols, and error descriptions are described. Only a debug-type OS returns return codes. If using C language, return codes are returned as system call return values. If using assembly language, they are returned to the C register. However, event flag-related system call return codes are returned to variables specified by parameters when using C language.
8. Assembler format. This is the format used to call a system call from assembly language. In the assembler format, system calls are macro-defined, so only the macro name and parameters are entered here.

7.3 Task-related System Calls

This section describes task-related system calls, which are listed below.

sta_tsk(),	sta_tsk1(),	sta_tsk2(),	sta_tsk3(),
sta_tske(),	sta_tsk1e(),	sta_tsk2e(),	sta_tsk3e(),
sta_tskp(),	sta_tskpe(),	ter_tsk(),	sta_tskt(),
sta_tskte(),	chg_tskt(),	chg_tskte(),	ter_tskt(),
sta_tskf(),	ter_tskf(),		
rot_rdq(),	rot_rdq1(),	rot_rdq2(),	rot_rdq3(),
rot_rdqe(),	rot_rdq1e(),	rot_rdq2e(),	rot_rdq3e(),
tsk_sts(),	chg_pri(),	ext_tsk(),	

7.3.1 sta_tsk

sta_tsk	STArt TaSK not called in ter_tsk or tsk_sts

[Function]

This system call chains the specified task to the end of the ready queue. However, this system call should only be used when the specified task has not been specified as a parameter for either ter_tsk() or tsk_sts(), or when the parameter specified for ter_tsk() and tsk_sts() is not explicit.

[Format (C)]

Slim type: sta_tsk(a_tsk);

Debug type: unsigned char err;
err = sta_tsk(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and “TE_QOVR” is returned if the area for registering the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tsk a_tsk

7.3.2 sta_tsk1

sta_tsk1	STArt TaSK called in ter_tsk

[Function]

This system call chains the specified task to the end of the ready queue. However, this system call should only be used when the specified task has been specified as a parameter and for ter_tsk() but not for tsk_sts().

[Format (C)]

Slim type: sta_tsk1(a_tsk);

Debug type: unsigned char err;
err = sta_tsk1(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. In addition, it performs setup for high-speed searching when the ter_tsk() system call is issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and “TE_QOVR” is returned if the area for registering the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tsk1 a_tsk

7.3.3 sta_tsk2

sta_tsk2	STArt TaSK called in tsk_sts

[Function]

This system call chains the specified task to the end of the ready queue. However, this system call should only be used when the specified task has been specified as a parameter for tsk_sts() but not for ter_tsk().

[Format (C)]

Slim type: sta_tsk2(a_tsk);

Debug type: unsigned char err;
err = sta_tsk2(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and changes it from DORMANT status to READY status. In addition, it performs setup for high-speed searching when the tsk_sts() system call is issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tsk2 a_tsk

7.3.4 sta_tsk3

sta_tsk3	STArt TaSK called in tsk_sts and tsk_sts

[Function]

This system call chains the specified task to the end of the ready queue. However, this system call should only be used when the specified task has been specified as a parameter for both ter_tsk() and tsk_sts().

[Format (C)]

Slim type: sta_tsk3(a_tsk);

Debug type: unsigned char err;
err = sta_tsk3(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. In addition, it performs setup for high-speed searching when the ter_tsk() and tsk_sts() system calls are issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tsk3 a_tsk

7.3.5 sta_tske

sta_tske	STArt TaSK not called in ter_tsk or tsk_sts, and Exit task

[Function]

This system call chains the specified task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the specified task has not been specified as a parameter for either ter_sts() or tsk_sts() or when the parameter specification for ter_tsk() or tsk_sts() is not explicit.

[Format (C)]

```
sta_tske(a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. At the same time, it changes the current task to DORMANT status.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tske  a_tsk
```

7.3.6 sta_tsk1e

sta_tsk1e	STArt TaSK called in ter_tsk, and Exit task

[Function]

This system call chains the specified task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the specified task has been specified as a parameter for ter_tsk() but not for tsk_sts().

[Format (C)]

```
sta_tsk1e(a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. At the same time, it changes the current task to DORMANT status and performs setup for high-speed searching when the ter_tsk() system call is issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tsk1e  a_tsk
```

7.3.7 sta_tsk2e

sta_tsk2e	STArt TaSK called in tsk_sts, and Exit task

[Function]

This system call chains the specified task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the specified task has been specified as a parameter for tsk_sts() but not for ter_tsk().

[Format (C)]

```
sta_tsk2e(a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. At the same time, it changes the current task to DORMANT status and performs setup for high-speed searching when the tsk_sts() system call is issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tsk2e  a_tsk
```

7.3.8 sta_tsk3e

sta_tsk3e	STArt TaSK called in ter_tsk and tsk_sts, and Exit task

[Function]

This system call chains the specified task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the specified task has been specified as a parameter for both tsk_sts() and ter_tsk().

[Format (C)]

```
sta_tsk3e(a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status. At the same time, it changes the current task to DORMANT status and performs setup for high-speed searching when the ter_tsk() and tsk_sts() system calls are issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tsk3e  a_tsk
```

7.3.9 sta_tskp

sta_tskp	STArt TaSK and change Priority

[Function]

This system call chains the specified task to the start of the ready queue.

[Format (C)]

Slim type: sta_tskp(a_tsk);

Debug type: unsigned char err;
err = sta_tskp(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the end of the ready queue and then changes it from DORMANT status to READY status.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the ready queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

sta_tskp a_tsk

7.3.10 sta_tskpe

sta_tskpe	STArt TaSK and change Priority, and Exit task

[Function]

This system call chains the specified task to the start of the ready queue and then terminates the current task.

[Format (C)]

```
sta_tskpe(a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started

[Description]

This system call chains the specified task “a_tsk” to the start of the ready queue and then changes it from DORMANT status to READY status. At the same time, it changes the current task to DORMANT status.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tskpe  a_tsk
```

7.3.11 ter_tsk

ter_tsk	TERminate TaSK in ready queue

[Function]

This system call forcibly terminates a specified task that has been chained to the ready queue.

[Format (C)]

Slim type: ter_tsk(a_tsk);

Debug type: unsigned char err;
err = ter_tsk(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be terminated

[Description]

This system call removes the specified task “a_tsk” from the ready queue and then changes the specified task to DORMANT status.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was removed normally from the ready queue and the return code “TE_DMT” is returned if the specified task was not chained to the ready queue.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x0d	TE_DMT	Specified task is in DORMANT status

[Format (Assembler)]

ter_tsk a_tsk

7.3.12 sta_tskt

sta_tskt	STArt TaSK after a time

[Function]

This system call chains the specified task to the start of the ready queue after a specified amount of time has elapsed.

[Format (C)]

Slim type: sta_tskt(a_tsk,tmout);

Debug type: unsigned char err;
err = sta_tskt(a_tsk,tmout);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started
tmout	16	unsigned int	Time until task is started

[Description]

This system call chains the specified task “a_tsk” to the timer queue for the amount of time indicated by the “tmout” parameter, after which it places “a_tsk” at the start of the ready queue.

The time specified by the “tmout” should be set as the interval time for timer interrupts.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task is chained normally to the timer queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the timer queue.

After the specified time has elapsed, during operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (timer queue)

[Format (Assembler)]

sta_tskt a_tsk,tmout

7.3.13 sta_tskte

sta_tskte	STArt TaSK after a time, and Exit task

[Function]

This system call chains the specified task to the start of the ready queue after a specified amount of time has elapsed, then terminates the current task.

[Format (C)]

sta_tskte(a_tsk,tmout)

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started
tmout	16	unsigned int	Time until task is started

[Description]

This system call chains the specified task “a_tsk” to the timer queue for the amount of time indicated by the “tmout” parameter and the current task is changed to DORMANT status. After the specified time has elapsed, “a_tsk” is chained to the start of the ready queue.

The time specified by the “tmout” should be set as the interval time for timer interrupts.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

After the specified time has elapsed, during operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
sta_tskte  a_tsk,tmout
```

7.3.14 chg_tskt

chg_tskt	CHanG TaSK's startTime

[Function]

This system call changes the startup time for the specified task.

[Format (C)]

Slim type: chg_tskt(a_tsk,tmout);

Debug type: unsigned char err;
err = chg_tskt(a_tsk,tmout);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started
tmout	16	unsigned int	Time until task is started

[Description]

The startup time for the specified task "a_tsk" is changed to the specified time "tmout". When this system call is issued, it searches first in the timer queue and then in the ready queue to determine whether or not the specified task already exists. If it does exist, this system call forcibly terminates the task and chains it again to the timer queue. The specified task is also chained to the timer queue when it is not found in the ready queue or timer queue.

The time specified by the "tmout" should be set as the interval time for timer interrupts.

When using a debug-type OS, the return code "TE_OK" is returned if the specified task was chained normally to the timer queue and the return code "TE_QOVR" is returned if the area for chaining the specified task does not exist in the timer queue.

After the specified time has elapsed, during operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (timer queue)

[Format (Assembler)]

chg_tskt a_tsk,tmout

7.3.15 chg_tskte

chg_tskte	CHanG TaSK's startTime, and Exit task

[Function]

This system call changes the startup time for the specified task and then terminates the current task.

[Format (C)]

```
chg_tskte(a_tsk,tmout)
```

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be started
tmout	16	unsigned int	Time until task is started

[Description]

The startup time for the specified task "a_tsk" is changed to the specified time "tmout" and then the current task is terminated. When this system call is issued, it searches first the timer queue and then in the ready queue to determine whether or not the specified task already exists. If it does exist, this system call forcibly terminates the task and chains it again to the timer queue. The specified task is also chained to the timer queue when it is not found in the ready queue or timer queue.

The time specified by the "tmout" should be set as the interval time for timer interrupts.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

After the specified time has elapsed, during operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
chg_tskte  a_tsk,tmout
```

7.3.16 ter_tskt

ter_tskt	TERminate TaSK in ready queue or Timer queue

[Function]

This system call forcibly terminates the specified task, which has been chained to the timer queue or ready queue.

[Format (C)]

Slim type: ter_tskt(a_tsk);

Debug type: unsigned char err;
err = ter_tskt(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be terminated

[Description]

This system call removes the specified task “a_tsk” from the timer queue or ready queue and then changes the specified task to DORMANT status. If the specified task did not have a specified wait time task, it is still removed from the ready queue if it has been chained there.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was removed normally from the timer queue or ready queue and the return code “TE_DMT” is returned if the specified task was not chained to either the timer queue or the ready queue.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x0d	TE_DMT	Specified task was not chained

[Format (Assembler)]

ter_tskt a_tsk

7.3.17 sta_tskf

sta_tskf	STArt TaSK with event-Flag

[Function]

This system call registers a task to be started by an event flag operation.

[Format (C)]

```
sta_tskf(a_flg,pk_finfo);
```

[Parameters]

Parameter	Bits	Type	Description
a_flg	16	Function	Address of target event flag
pk_finfo	8	char	Event flag auxiliary data ID number

Parameter in auxiliary data	Bits	Type	Description
waiptn	8	char	Task's wait bit pattern
wfmode	8	char	Task's start condition
a_tsk	16	Function	Start address of task to be started

[Description]

This system call registers the task specified by an ID number in the event flag auxiliary data "pk_finfo" to event flag "a_flg". The ID number specified by the "pk_finfo" parameter is an ordinal number (1, 2, 3 …) that begins at the start of the data in the event flag data table.

Event flag auxiliary data must exist in either ROM or RAM within a table called "_MX_flg_TBL". For a description of how event flag data tables are created, see "**10.3.6 Allocation of event flag**".

Specify "EVENT_AND" if the task's start condition is an AND condition or "EVENT_OR" if it is an OR condition.

[Return codes]

None

[Format (Assembler)]

```
sta_tskf  a_flg,pk_finfo
```

7.3.18 rot_rdq

rot_rdq	ROTate ReaDy Queue (current task is not called in ter_tsk or tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue. However, this system call should only be used when the current task has not been specified as a parameter for either ter_tsk() or tsk_sts(), or when the parameter specification for ter_tsk() and tsk_sts() is not explicit.

[Format (C)]

Slim type: rot_rdq();

Debug type: unsigned char err;
err = rot_rdq();

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was chained normally to the end of the ready queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of the current task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

rot_rdq

7.3.19 rot_rdq1

rot_rdq1	ROTate ReaDy Queue (current task is called in ter_tsk)

[Function]

This system call chains the current task to the end of the ready queue. However, this system call should only be used when the current task has been specified as the parameter for ter_tsk() but not for tsk_sts().

[Format (C)]

Slim type: rot_rdq1();

Debug type:
```
unsigned char err;
err = rot_rdq1();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue. It also performs setup for high-speed searching when the ter_tsk()system call is issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was chained normally to the end of ready queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of the current task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

```
rot_rdq1
```

7.3.20 rot_rdq2

rot_rdq2	ROTate ReaDy Queue (current task is called in tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue. However, this system call should only be used when the current task has been specified as a parameter for tsk_sts() but not for ter_tsk().

[Format (C)]

Slim type: rot_rdq2();

Debug type:
```
unsigned char err;
err = rot_rdq2();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue. It also performs setup for high-speed searching when the tsk_sts() system call is issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was chained normally to the end of the ready queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of the current task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

rot_rdq2

7.3.21 rot_rdq3

rot_rdq3	ROTate ReaDy Queue (current task is called in ter_tsk and tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue. However, this system call should only be used when the current task has been specified as a parameter for both ter_tsk() and tsk_sts().

[Format (C)]

Slim type: rot_rdq3();

Debug type:
```
unsigned char err;
err = rot_rdq3();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue. It also performs setup for high-speed searching when the ter_tsk() or tsk_sts system call is issued.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was chained normally to the end of the ready queue and the return code “TE_QOVR” is returned if the area for chaining the specified task does not exist in the ready queue.

During operation of the current task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)

[Format (Assembler)]

```
rot_rdq3
```

7.3.22 rot_rdqe

rot_rdqe	ROTate ReaDy Queue and Exit task (current task is called in ter_tsk and tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the current task has not been specified as a parameter for either ter_tsk() or tsk_sts() or when the parameter specification for ter_tsk() or tsk_sts() is not explicit.

[Format (C)]

```
rot_rdqe();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue and then changes the current task to DORMANT status.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
rot_rdqe
```

7.3.23 rot_rdq1e

rot_rdq1e	ROTate ReaDy Queue and Exit task (current task is called in ter_tsk)

[Function]

This system call chains the current task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the current task has been specified as a parameter for ter_tsk() but not for tsk_sts().

[Format (C)]

```
rot_rdq1e();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue and then changes the current task to DORMANT status. It also performs setup for high-speed searching when the ter_tsk() system call is issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
rot_rdq1e
```

7.3.24 rot_rdq2e

rot_rdq2e	ROTate ReaDy Queue and Exit task (current task is called in tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the current task has been specified as a parameter for tsk_sts() but not for ter_tsk().

[Format (C)]

```
rot_rdq2e();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue and then changes the current task to DORMANT status. It also performs setup for high-speed searching when the tsk_sts() system call is issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
rot_rdq2e
```

7.3.25 rot_rdq3e

rot_rdq3e	ROTate ReaDy Queue and Exit task (current task is called in ter_tsk and tsk_sts)

[Function]

This system call chains the current task to the end of the ready queue and then terminates the current task. However, this system call should only be used when the current task has been specified as a parameter for both ter_tsk() and tsk_sts().

[Format (C)]

```
rot_rdq3e();
```

[Parameters]

None

[Description]

This system call chains the current task to the end of the ready queue and then changes the current task to DORMANT status. It also performs setup for high-speed searching when the ter_tsk() or tsk_sts() system call is issued.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task's operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

None

[Format (Assembler)]

```
rot_rdq3e
```

7.3.26 tsk_sts

tsk_sts	get TaSK STatuS

[Function]

The system call checks the status of a task.

[Format (C)]

```
tsk_sts(pk_tskst,a_tsk);
```

[Parameters]

Parameter	Bits	Type	Description
pk_tskst	16	Function	Area for storing task status
a_tsk	16	Function	Start address of task to be checked

[Description]

If the specified task “a_tsk” has been chained to the ready queue, this system call returns “TTS_RDY” to the “pk_tskst” area. Otherwise, it returns “TTS_DMT” to the “pk_tskst” area.

[Return codes]

Return code	Symbol	Cause
0x02	TTS_RDY	Specified task is in READY status
0x10	TTS_DMT	Specified task is in DORMANT status

[Format (Assembler)]

```
tsk_sts  pk_tskst,a_tsk
```

7.3.27 chg_pri

chg_pri	CHanGe PRIority

[Function]

This system call again chains the specified task to the start of the ready queue.

[Format (C)]

Slim type: chg_pri(a_tsk);

Debug type: unsigned char err;
err = chg_pri(a_tsk);

[Parameters]

Parameter	Bits	Type	Description
a_tsk	16	Function	Start address of task to be checked

[Description]

If the specified task “a_tsk” has already been chained to the ready queue, this system call chains it again to the start of the ready queue.

When using a debug-type OS, the return code “TE_OK” is returned if the specified task was already chained to the ready queue and the return code “TE_DMT” if it has not been chained there.

During operation of a task that has been chained to the ready queue, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x0d	TE_DMT	Specified task is in DORMANT status

[Format (Assembler)]

chg_pri a_tsk

7.3.28 ext_tsk

ext_tsk	Exit task

[Function]

This system call terminates the current task.

[Format (C)]

```
ext_tsk();
```

[Parameters]

None

[Description]

This system call sets the current task to the DORMANT status.

When using a debug-type OS, processing is terminated after this system call is issued, so no return codes are returned.

[Return codes]

None

[Format (Assembler)]

```
ext_tsk
```

7.4 Event Flag-related System Calls

This section describes event flag-related system calls, which are listed below.

set_flg(), clr_flg(), rpl_flg()

7.4.1 set_flg

set_flg	SET event-FLaG

[Function]

This system call sets any bit pattern to the specified event flag.

[Format (C)]

Slim type: set_flg(a_flg,setptn);

Debug type: unsigned char err;
set_flg(a_flg,setptn,err);

[Parameters]

Parameter	Bits	Type	Description
a_flg	16	Function	Address of specified event flag
setptn	8	char	Bit pattern to be set
err	8	unsigned char	Area for storing return codes (debug type only)

[Description]

This system call sets OR data for the bit pattern of the specified event flag “a_flg” and the set pattern “setptn” to the specified event flag. Specify “1” as the bit value to be set to the set pattern “setptn”.

If the start condition of the task registered to the specified event flag is met, the registered task is chained to the start of the ready queue.

When using a debug-type OS, the return code “TE_OK” is returned if the registered task was started by setting the event flag. The return code “TE_QOVR” is returned if the start condition was met but the task could not be chained to the ready queue because the area for chaining the specified task does not exist in the ready queue. The return code “TE_SET” is returned if the event flag was successfully set and the return code “TE_NOENT” is returned if the task was not registered to the specified event flag.

During operation of a task that has been chained to the ready queue after the start condition was met, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)
0x04	TE_SET	Event flag was successfully set
0x05	TE_NOENT	Task to be started was not registered

[Format (Assembler)]

set_flg a_tsk,setptn

7.4.2 clr_flg

clr_flg	CLeaR event-FLaG

[Function]

This system call clears any bit pattern from the specified event flag.

[Format (C)]

```
clr_flg(a_flg,clrptn);
```

[Parameters]

Parameter	Bits	Type	Description
a_flg	16	Function	Address of specified event flag
clrptn	8	char	Bit pattern to be cleared

[Description]

This system call clears the bit pattern specified by the clear pattern “clrptn” from the specified event flag “a_flg”. Specify “0” as the bit value of the clear pattern “clrptn” (or set “1” to avoid clearing). Once the event flag has been cleared, the registered task will not be started even when its start condition is met.

[Return codes]

None

[Format (Assembler)]

```
clr_flg  a_flg,clrptn
```

7.4.3 rpl_flg

rpl_flg	RePLace event-FLaG

[Function]

This system call replaces the bit pattern of the specified event flag with the specified bit pattern.

[Format (C)]

Slim type: rpl_flg(a_flg,rplptn);

Debug type: unsigned char err;
rpl_flg(a_flg,rplptn,err);

[Parameters]

Parameter	Bits	Type	Description
a_flg	16	Function	Address of specified event flag
rplptn	8	char	Bit pattern to be set as replacement
err	8	unsigned char	Area for storing return codes (debug type only)

[Description]

This system call replaces the bit pattern of the specified event flag “a_flg” with the specified bit pattern “rplptn”.

If the start condition of the task registered to the specified event flag is met, the registered task is chained to the start of the ready queue.

When using a debug-type OS, the return code “TE_OK” is returned if the registered task was started by replacing the event flag. The return code “TE_QOVR” is returned if the start condition was met but the task could not be chained to the ready queue because the area for chaining the specified task does not exist in the ready queue. The return code “TE_SET” is returned if the event flag was successfully replaced and the return code “TE_NOENT” is returned if the task was not registered to the specified event flag.

During operation of a task that has been chained to the ready queue after the start condition was met, the interrupt enable/prohibit status of the previous task’s operation is continued, so be sure to set the interrupt status for each task.

[Return codes]

Return code	Symbol	Cause
0x00	TE_OK	Normal termination
0x11	TE_QOVR	Overflow (ready queue)
0x04	TE_SET	Event flag was successfully replaced
0x05	TE_NOENT	Task to be started was not registered

[Format (Assembler)]

rpl_flg a_tsk,rplptn

CHAPTER 8 THIS OS'S MANAGEMENT AREA

8.1 This OS's Memory Area

The memory area used by this OS is described below.

- Code size
  - Slim type: approx. 1.7 Kbytes max.
  - Debug type: approx. 1.8 Kbytes max.

- Data size
  - Ready queue
    4 × (maximum number of tasks simultaneously chained to ready queue plus 1) bytes

  - Timer queue
    4 × (maximum number of tasks simultaneously chained to timer queue plus 1) bytes

  - Event flag
    2 × (number of event flags) + 1 byte

  - Event flag auxiliary data
    4 × (number of event flag auxiliary data units) bytes

  - System management area (saddr area)
    8 bytes

8.2 How to Estimate Memory Size

The method for calculating the amount of RAM used by this OS is described below.

(Example)

- Number of tasks : 40 (of which 13 use the timer)
- Number of event flags : 10
- Number of event flag auxiliary data units : 5 (defined in RAM)

* Although there are 40 tasks, only 15 at most can be chained to the ready queue at one time.
* Although there are 13 tasks that use the timer, only 9 at most can be chained to the timer queue at one time.

Ready queue size	: 4 × (15 tasks + 1) bytes	= 64 bytes
Timer queue size	: 4 × (9 tasks + 1) bytes	= 40 bytes
Event flag table size	: 2 × 10 event flags' bytes	= 20 bytes
Event flag data table size	: 4 × 5 event flags' bytes	= 20 bytes
System management area	:	8 bytes
Total		152 bytes

In this system example, a total of 152 bytes of RAM is required.

[MEMO]

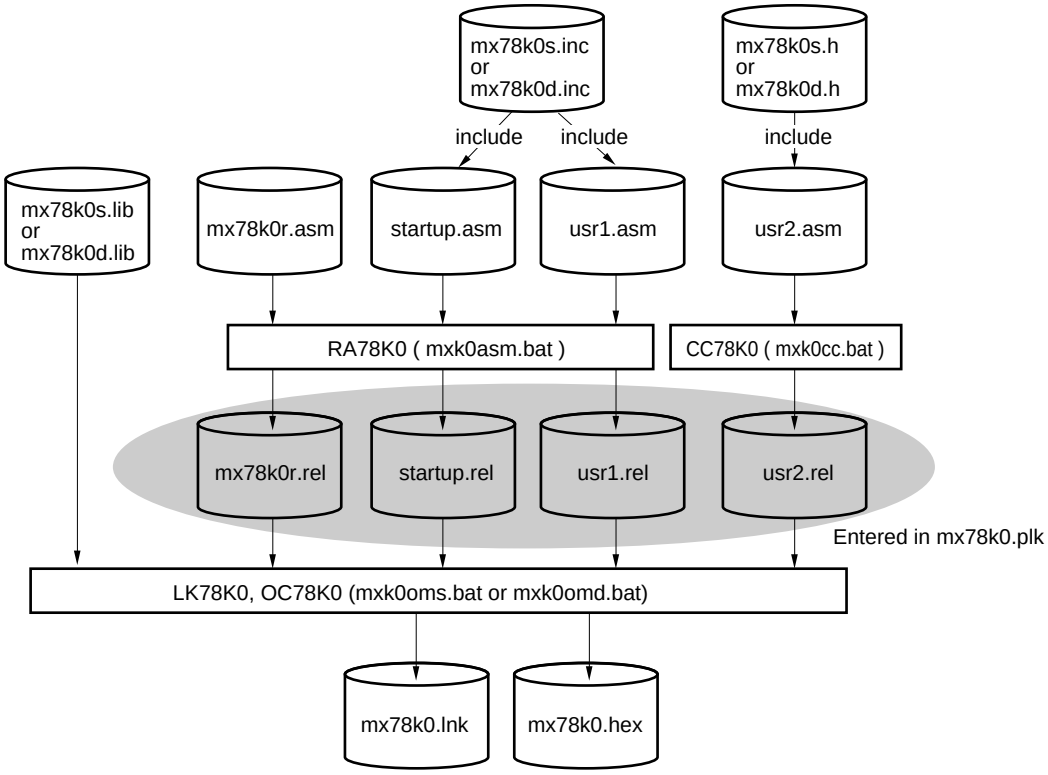
CHAPTER 9 APPLICATION DEVELOPMENT STEPS

This chapter describes the steps for creating user applications, related cautions, and initialization of this OS.

9.1 Steps in Creation of Load Module

The steps used when creating a load module are shown in Figure 9-1.

Figure 9-1. Steps in Creation of Load Module



The various files shown in Figure 9-1 are described below. (Names of files that are bundled with the product are underlined. This includes files that are bundled as samples.)

mx78k0s.inc	:	Slim type include file (assembly language)
mx78k0d.inc	:	Debug type include file (assembly language)
mx78k0s.h	:	Slim type include file (C language)
mx78k0d.h	:	Debug type include file (C language)
mx78k0s.lib	:	Slim type system call library file
mx78k0d.lib	:	Debug type system call library file
mx78k0r.asm	:	Memory definition file
startup.asm	:	Startup routine
usr1.asm	:	User task (assembly language)
usr2.c	:	User task (C language)
mx78k0.plk	:	Linker parameter file
mx78k0.lnk	:	Load module (not including data in ROM)
mx78k0.hex	:	Load module (including data in ROM)

9.2 Cautions on Load Module Creation

The following describes caution points and options regarding use of the provided batch files for the make function.

1. Batch files

- The batch file for starting the assembler (mxk0asm.bat) and the batch file for starting the compiler (mxk0cc.bat) have no particular startup location. The files (having the extension ".rel") that are output by these batch files are created in the current directory.
- When using a batch file for creating a load module, the linker parameter file (mx78k0.plk) is placed in the same directory as the linked file group (also having the extension ".rel"). Start the batch file from the directory where these files reside.
- If the system call library file was installed in a directory other than the default directory (\nectools\lib78k0), be sure to change the LK78K0 option that is entered in the batch file for creating a load module.

2. Assembler (RA78K0) options

- Use only the "-s -nca" options among the RA78K0 options. Assembly and link operations may not occur normally if any option that invalidates these options is specified.

3. Compiler (CC78K0) options

- Use only the "-qcr -s -nca" options among the CC78K0 options. Compile and link operations may not occur normally if any option that invalidates these options is specified.
- Do not use the "-za" option.
- Do not use the "-zo" option.

4. Linker (LK78K0) options

- If the "-s" option is removed from the LK78K0 options, the stack pointer will no longer be created automatically. If the "-s" option has been removed, use a memory definition file ("mx78k0r.asm") to define a start address ("_@STBEG") for the stack pointer.

9.3 Cautions on Creating User Tasks

This section describes caution points regarding the creation of user tasks. For description of how the various system calls are used (their format, etc.), see "**CHAPTER 7 SYSTEM CALLS**".

9.3.1 When coding a task in assembly language

Example of user task description (assembly language)

```
; User Task Sample (usr1.asm)
NAME    USR1
$INCLUDE(MX78KOS.INC)           <1>
EXTRN   n_500ms                 <2>
EXTRN   _task3
PUBLIC  _task1
PUBLIC  _task2
        CSEG
_task1: CLR1    P1.0
        sta_tsk _task2
        sta_tskt _task3,n_500ms  <2>
        ext_tsk                 <3>
_task2: BT      P0.0,$label1
        rot_rdqe                <3>
label1: ext_tsk                 <3>
        END
```

<1> Use an include file that is appropriate for the system call type being used.
(Slim type is used in the above example.)

<2> When using a system call (such as sta_tskt or chg_tskt) that takes a time value from a parameter, the parameter's time specification should be calculated based on the timer interrupt interval time being used by the OS. (Example: If the interval time is 10 ms, then define n_500 ms = 50.)

<3> For the final task (such as ext_tsk), enter a system call that is used to terminate the current task.

<4> When using a debug type system call, the system call's return codes are stored in the C register.

9.3.2 When coding a task in C language

Example of user task description (C language)

```
/* User Task Sample (usr2.c) */
#pragma SFR
#include “mx78k0s.h”                        <1>
extern int n_10ms;                          <2>
extern void task1();
extern void task2();
void task3();
void task4();

void task3();
{
        unsigned char pk_tskst;
        tsk_sts(&pk_tskst,task2);
        if (pk_tskst == TTS_RDY)
        {
                ter_tsk(task2);
                P1.0 = 1;
                sta_tskte(task4,n_10ms);   <2><3>
        }
        sta_tske(task1);                    <3>
}
void task4()
{
        P1.0 = 0;
        sta_tske(task1);                    <3>
}
```

<1> Use an include file that is appropriate for the system call type being used.
(Slim type is used in the above example.)

<2> When using a system call (such as sta_tskt or chg_tskt) that takes a time value from a parameter, the parameter’s time specification should be calculated based on the timer interrupt interval time being used by the OS.

<3> For the final task (such as ext_tsk), enter a system call that is used to terminate the current task.

<4> When using an event flag, set the type declarations and terminations in the event flag table and the auxiliary data table. For details, see **“10.3.6 Allocation of event flag”**.

<5> When using a debug type system call, the system call’s return codes are returned as the system call return value (unsigned char type).

9.4 User's Own Coding

The following describes parts other than user tasks that the user must create.

9.4.1 Initialization Process

This process is used to initialize the hardware and software during system startup. For details of system initialization process, see "**CHAPTER 10 SYSTEM INITIALIZATION PROCESS**".

9.4.2 Timer Process

When using a system call that uses the timer queue, a timer handler and timer interrupt prohibit/enable routines must be created.

1. Timer handler

This describes the interrupt servicing for the timer used by the OS. As part of initialization processing, allocate one timer for exclusive use by the OS and set an interrupt vector. (For details, see the User's Manual for the target device.)
As part of interrupt servicing, be sure to enter an instruction that calls the OS's timer processing routine ("_MX_int").

(Example) When start address of timer interrupt handler is "TM1INT"

```
TM1INT:
        CALL    !_MX_int
        RETI
```

Note Do not enter any instruction other than an instruction that calls the OS's timer processing routine ("_MX_int") within the timer interrupt handler. If a user program is entered within the handler, the OS's timer processing and the user program may not operate normally due to the effects of multiple interrupts, etc.

2. Timer interrupt prohibit routine

Create a routine to prohibit the timer interrupts used by the OS. Use "Prohibit_MX_int" as the start address for processing and use the "RET" instruction to terminate.

(Example) When using TM1

```
_Prohibit_MX_int:
        PUSH    PSW
        DI
        SET1    MK0H.1          ;  TMMK1←1
        POP     PSW
        RET
```

3. Timer interrupt enable routine

Create a routine to enable the timer interrupts used by the OS. Use “Permit_MX_int” as the start address for processing and use the “RET” instruction to terminate.

(Example) When using TM1

```
_Permit_MX_ int:
        PUSH    PSW
        DI
        CLR1    MK0H.1          ;  TMMK1←0
        POP     PSW
        RET
```

9.4.3 Cautions on coding tasks

Do not combine the register banks used by tasks with those used by the OS. Operations cannot be guaranteed if the OS and the tasks share the same register banks.

9.4.4 Cautions on coding the interrupt handler

Only some system calls can be called by the interrupt handler. If the issued system call is not one of these types, the application may not operate normally.

User the “RETI” instruction to terminate the interrupt handler code description.

Do not combine the register banks used by the interrupt handler with those used by the OS. Operations cannot be guaranteed if the OS and the interrupt handler share the same register banks.

CHAPTER 10 SYSTEM INITIALIZATION PROCESS

This chapter describes the initialization of hardware and this OS's system area during system start-up. System initialization processing includes a part that initializes the hardware and a part that initializes the software.

10.1 Overview of System Initialization Process

The system initialization process includes the following topics.

Table 10-1. Example of Initialization Processing

	Contents of process
Hardware	CPU initialization • Ports • Timer • Interrupts • RAM Initialization of peripheral devices
Software	Allocation of areas used by OS • Set stack pointer • Set register banks • Allocate queue • Allocate system work area • Allocate event flag • Allocate area for "tsk_sts" OS initialization • Queue initialization • Set event flag • Start initial task

10.2 Hardware Initialization

For hardware initialization, be sure to initialize all of the hardware needed to operate the user system. (See the various devices' User's Manuals for details of hardware initialization.)

If using a timer-related system call, be sure to allocate one timer for use by the OS.

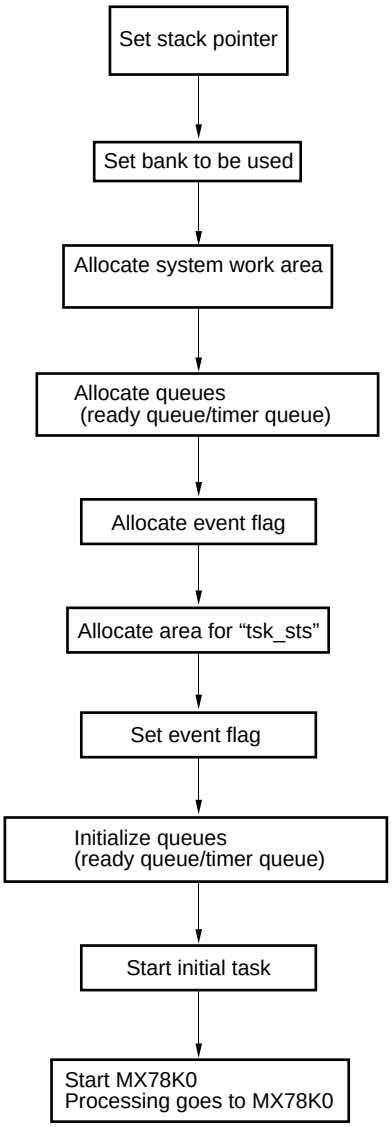
No code samples for the part that initializes hardware are provided, so users may need to create their own samples.

10.3 Software Initialization

Software initialization includes allocating and initializing the area used by the OS and then starting the initial task. One method for software initialization is to simply revise the provided sample memory definition file (“mx78k0r.asm”). (In the descriptions shown below, the user-defined parts are indicated by “xx”. Other parts are already entered in the sample memory definition file and do not need to be changed. In the sample code, PUBLIC declarations regarding the OS have been kept to a minimum, so more may need to be added depending on the actual user system. EXTRN declarations have been omitted completely.)

The software initialization steps are shown below.

Figure 10-1. Software Initialization Steps



10.3.1 Setting of stack pointer

This OS uses a stack pointer that is automatically created by the linker. When using one of the provided batch files for the make function, the stack pointer is automatically created, so there is no need to set it as part of the initialization process. However, if the "-s" option has been removed as a linker option, add the following to the memory definition file to set a stack pointer ("_@STBEG").

```
PUBLIC  _@STBEG
_@STBEG EQU     xxxxH   ; Start address of stack pointer
```

10.3.2 Setting of register banks

During its operations, this OS makes exclusive use of one of the register banks. In the memory definition file, set the register bank's value based on the following table.

Table 10-2. Register Bank Values

Register bank to be used	Setting
Bank 0	0D061H
Bank 1	0D861H
Bank 2	0F061H
Bank 3	0F861H

```
PUBLIC  _MX_Bank
_MX_BankEQU     xxxxH   ; Register bank setting
```

10.3.3 Allocation of system work area

Allocate the system work area (SWA) to be used by the OS. To allocate this area, set the SWA's start address in the memory definition file. The SWA must be allocated in a short direct addressing area.

```
PUBLIC          _MX_ctask, _MX_tq_s, _MX_tqv_s
PUBLIC          _MX_ter_mem, _MX_sts_mem, az_arg
_MX78K0_WORK    EQU     xxxxH           ; Start address of SWA2
_MX_SWA2        DSEG    AT _MX78K0_WORK
_MX_ctask:      DS      2
_MX_tq_s:       DS      1
_MX_tqv_s:      DS      1
_MX_ter_mem:    DS      1
_MX_sts_mem:    DS      1
az_arg:         DS      2
```

10.3.4 Allocation of ready queue

After setting the start address of RAM allocated for the ready queue and setting the maximum number of tasks that can be chained to the ready queue at the same time, allocate an area for the ready queue. The specification range for the maximum number of tasks that can be chained to the ready queue at the same time is from 1 to 63 tasks.

```
PUBLIC          _MX_rq_StartAddress
PUBLIC          _MX_rq_e0
_MX_rq_StartAddress     EQU         xxxxH       ; Start address of ready queue
_MX_rq_Max              EQU         xx          ; Maximum number of tasks
mx78k0rq                DSEG        _MX_rq_Start_Address
_MX_rq_area:            DS          (_MX_rq_Max + 1) * 4
_MX_rq_e0               EQU         _MX_rq_Max*4
```

Note Allocate a contiguous area in RAM as the area to be used by the ready queue.

10.3.5 Allocation of timer queue

After setting the start address of RAM allocated for the ready queue, set the maximum number of tasks that can be chained to the timer queue at the same time. Note that there is no need to set up a timer queue if timer-related system calls will not be used.

The specification range for the maximum number of tasks that can be chained to the timer queue is from 1 to 62 tasks.

```
PUBLIC          _MX_tq_StartAddress
PUBLIC          _MX_tq_a_e0
PUBLIC          _MX_tq_r_e0
PUBLIC          _MX_tq_half
_MX_tq_StartAddress     EQU         xxxxH       ; Start address of timer queue
_MX_tq_Max              EQU         xx          Maximum number of tasks
mx78k0tq                DSEG        _MX_tq_Start_Address
_MX_tq_area:            DS          (_MX_tq_Max + 1) * 4
_MX_tq_r_e0             EQU         _MX_rq_Max*4
_MX_tq_a_e0             EQU         _MX_tq_StartAddress + _MX_tq_r_e0
_MX_tq_half             EQU         ((_MX_tq_Max + 1) SHR 1) * 4
```

Note Allocate a contiguous area in RAM as the area to be used by the timer queue.

10.3.6 Allocation of event flag

Allocate areas for an event flag table and an event flag data table. Note that there is no need to set up these tables if event flag-related system calls will not be used. The event flag table should be allocated in a contiguous area of RAM. Allocate a one-byte area at the end of the event flag table and set “0xff” to this area before using any event flags. The event flag data table can be set up in either ROM or RAM, but in either case the entire table must be set within a contiguous area. It is not always necessary to set up these table as part of the system initialization process.

1. When setting up a table using assembly language

```
PUBLIC          _MX_flags
PUBLIC          _MX_flg_tbl
        DSEG
_MX_flags:                      ; Event flag table
_flag1:         DS      2       ; Area for flag “_flag1”
_flag2:         DS      2       ; Area for flag “_flag2”
_flag3:         DS      2       ; Area for flag “_flag3”
_flag_end:      DS      1       ; Area for flag termination

                CSEG
_MX_flg_tbl:                    ; Event flag data table
        DB      EVENT_XXX       ; Wait mode (EVENT_OR, EVENT_AND)
        DB      xxxxxxxxB       ; Wait bit pattern
        DW      xxxx            ; Start address of task to be started
        DB      EVENT_XXX       ; Data volume setting
        DB      xxxxxxxxB
        DW      xxxx
```

Notes 1. The start label for the event flag table is set as “_MX_flags” and the start label for the event flag data table is set as “_MX_flg_tbl”. Do not omit or change these settings. All other labels are shown as examples only and can be freely changed by the user.

2. Be sure to set “0xff” at the end of the event flag table before using an event flag.

3. In the above example, the CSEG directive is used to allocate an event flag data table in ROM. When allocating in RAM, use the DSEG directive and set the auxiliary data as part of the initialization process. For details, see **“10.3.9 Set event flag”**.

2. When using C language to allocate

```
struct flaginfo                          /* Allocate event flag table */
{
       unsigned char flag1[2];           /* Allocate event flag */
       unsigned char flag2[2];
       unsigned char flag3[2];
       unsigned char flgend;             /* Flag termination symbol area */
} MX_flags;

struct event_tbl                         /* Type declaration for auxiliary data */
{
       const unsigned char wfmode;       /* Wait mode (EVENT_OR, EVENT_AND) */
       const unsigned char waiptn;       /* Wait bit pattern */
       const void (*func)(void);         /* Start address of task to be started */
};

const struct event_tbl MX_flg_tbl[2] =   /* Allocate event flag data table */
{                                        /* When using two event flag data tables */
       {EVENT_XXX, XX, XXXX}             /* {wfmode,waiptn,func} */
       {EVENT_XXX, XX, XXXX}
};
```

Notes 1. Use “MX_flags” as the variable name for the event flag table and “_MX_flg_tbl[ ]” as the variable name for the event flag data table. Do not change these variable names. All other member names for each table’s structure are shown as examples only and can be freely changed by the user.

2. As shown in the above example, an array containing a two-byte unsigned char type is used to enter each event flag in the event flag table. Normal operation may be impeded if an int type is used to allocate even the same two-byte area.

3. There is no need to set a type declaration for the event flag data table if the event flag data table is not being allocated using C language.

4. Be sure to set “0xff” at the end of the event flag table before using an event flag.

5. Define the event flag table and event flag data table as external variables.

10.3.7 Allocation of tsk_sts area

When using the tsk_sts() system call, allocate an area to be used for returning the task status. Specify the address of this allocated area as the first parameter in the tsk_sts() system call.

```
_pk_tskst               DS          1
```

Note In addition to allocating one area for the entire system via a setting in the memory definition file, another method is to use the tsk_sts() system call to allocate and use a one-byte variable for each task.

10.3.8 Initialization of queue

After allocating each area, initialize the queues. There are two queue initialization methods, depending on the queue configuration.

1. When using only the ready queue

Call the ready queue initialization routine.

```
    CALL        !_MX_init_rq
```

2. When using the ready queue and the timer queue

Call the ready queue/timer queue initialization routine.

```
    CALL        !_MX_init_tq
```

10.3.9 Setting of event flag

Initialize (clear to zero) the contents of the event flag table, then set the termination symbol (0xff). When allocating an event flag data table in RAM, set the event flag auxiliary data.

* In the example shown below, the structures of the event flag table and event flag data table are based on the code example shown in **"10.3.6 Allocation of event flag"** above.

1. When setting up a table using assembly language

```
        MOVW    AX,#00H
        MOVW    !_flag1,AX          ; Initialize contents of flag "_flag1"
        MOVW    !_flag2,AX          ; Initialize contents of flag "_flag2"
        MOVW    !_flag3,AX          ; Initialize contents of flag "_flag3"
        MOV     A, #0FFH
        MOV     !_flag_end,A        ; Set flag termination symbol (0xff)

        MOVW    AX,#_MX_flg_tbl     ; Set auxiliary data
        MOVW    HL,AX
        MOV     A,#EVENT_XXX        ; Wait mode for first flag
        MOV     [HL],A
        MOV     A,#xxxxxxxxB        ; Wait bit pattern
        MOV     [HL+1],A
        MOVW    AX,#xxxx            ; Startup task
        MOV     [HL+3],A
        XCH     A,X
        MOV     [HL+2],A

        MOV     A,#EVENT_XXX        ; Wait mode for second flag
        MOV     [HL+4],A
        MOV     A,#xxxxxxxxB        ; Wait bit pattern
        MOV     [HL+5],A
        MOVW    AX,#xxxx            ; Startup task
        MOV     [HL+7],A
        XCH     A,X
        MOV     [HL+6],A
```

2. When setting up a table using C language

```
void flag_init()
{
        MX_flags.flag1[0] = 0x00;               /* Clear contents of all flags */
        MX_flags.flag1[1] = 0x00;
        MX_flags.flag2[0] = 0x00;
        MX_flags.flag2[1] = 0x00;
        MX_flags.flag3[0] = 0x00;
        MX_flags.flag3[1] = 0x00;
        MX_flags.flgend = 0xff;                 /* Set flag termination symbol (0xff) */

                                                /* Set auxiliary data */
        MX_flg_tbl[0].wfmode = EVENT_XXX;       /* Wait mode for first flag */
        MX_flg_tbl[0].waiptn = XX;              /* Wait bit pattern*/
        MX_flg_tbl[0].func = XXXX;              /* Startup task */
        MX_flg_tbl[1].wfmode = EVENT_XXX;       /* Wait mode for second flag */
        MX_flg_tbl[1].waiptn = XX;              /* Wait bit pattern */
        MX_flg_tbl[1].func = XXXX;              /* Startup task */
}
```

10.3.10 Start of initial task

When the OS is started up, tasks that must be chained to the ready queue are started and are then chained to the ready queue.

```
    sta_tsk xxxx
    sta_tsk xxxx
```

10.3.11 OS startup

This passes control to the OS.

```
    BR      _MX_start
```

After the OS is started up, the task that has been chained to the start of the ready queue is executed.

[MEMO]

APPENDIX 1 LIST OF RESERVED WORDS

Reserved words are listed below.

az_arg,
chg_pri, chg_tskt, chg_tskte, clr_flg,
EVENT_AND, EVENT_OR, ext_tsk,
JudgeShift,
MEM_INVALID,
rot_rdq, rot_rdq1e, rot_rdq2e, rot_rdq3e, rot_rdqe, rot_rdq1e, rot_rdq2e, rot_rdq3e, rpl_flg,
set_flg, sta_tsk, sta_tsk1, sta_tsk2, sta_tsk3, sta_tske, sta_tsk1e, sta_tsk2e, sta_tsk3e,
sta_tskf, sta_tskp, sta_tskpe, sta_tskt, sta_tskte,
TE_DMT, TE_OK, TE_QOVR, ter_tsk, ter_tskf, ter_tskt, tsk_sts, TTS_DMT, TTS_RDY
_@STBEG,
_chg_pri, _chg_tskt, _chg_tskte,
_end_itdsp, _ext_tsk,
_MX_BACK, _MX_Bank, _MX_ctask, _MX_DeleteTask, _MX_end, _MX_end_dsp, MX_flags, _MX_flg_tbl,
_MX_FRONT, _MX_init_rq, _MX_init_tq, _MX_int, MX_rq_e0, _MX_rq_StartAddress, _MX_Search1,
_MX_Search2, _MX_start, _MX_sts_mem, _MX_ter_mem, _MX_tqv_s, _MX_tq_a_e0, _MX_tq_half,
_MX_tq_r_e0, _MX_tq_s, _MX_tq_StartAddress, _Permit_MX_int, _Prohibit_MX_int,
_rot_rdq, rot_rdq1e, rot_rdq2e, rot_rdq3e, _rot_rdq9, _rot_rdqe, _rot_rdq1e, _rot_rdq2e,
_rot_rdq3e, rot_rdq9e
_Shift1, _Shift2, _sta_tsk, _sta_tsk1, _sta_tsk2, _sta_tsk3, _sta_tsk9, _sta_tske, _sta_tsk1e
_sta_tsk2e, _sta_tsk3e, _sta_tsk9e, _sta_tskp, _sta_tskpe, _sta_tskt, _sta_tskte,
_ter_tsk, _ter_tskf, _ter_tskt, _tsetflg, _tsk_sts
TE_SET, TE_NOENT

[MEMO]

APPENDIX 2 LIST OF STACK SIZES USED BY SYSTEM CALLS

Stack sizes that are used within each system call are listed below.

Appendix 2. Stack Sizes

System call name	Stack sizes	
	Slim type	Debug type
sta_tsk	5 bytes	5 bytes
sta_tsk1	5 bytes	5 bytes
sta_tsk2	5 bytes	5 bytes
sta_tsk3	5 bytes	5 bytes
sta_tske	5 bytes	5 bytes
sta_tsk1e	5 bytes	5 bytes
sta_tsk2e	5 bytes	5 bytes
sta_tsk3e	5 bytes	5 bytes
sta_tskp	11 bytes	11 bytes
sta_tskpe	5 bytes	5 bytes
ter_tsk	10 bytes	10 bytes
sta_tskt	14 +α bytes	14 +α bytes
sta_tskte	16 +α bytes	16 +α bytes
chg_tskt	16 +α bytes	16 +α bytes
chg_tskte	16 +α bytes	16 +α bytes
ter_tskt	16 +α bytes	16 +α bytes
sta_tskf	0 byte	0 byte
ter_tskf	10 bytes	10 bytes
rot_rdq	3 bytes	3 bytes
rot_rdq1	3 bytes	3 bytes
rot_rdq2	3 bytes	3 bytes
rot_rdq3	3 bytes	3 bytes
rot_rdqe	3 bytes	3 bytes
rot_rdq1e	3 bytes	3 bytes
rot_rdq2e	3 bytes	3 bytes
rot_rdq3e	3 bytes	3 bytes
tsk_sts	12 bytes	12 bytes
chg_pri	10 bytes	10 bytes
ext_tsk	2 bytes	2 bytes
set_flg	15 bytes	15 bytes
clr_flg	0 byte	0 byte
rpl_flg	15 bytes	15 bytes
Timer interrupt servicing (_MX_int)	14 bytes	14 bytes
Ready queue initialization (_MX_init_rq)	3 bytes	3 bytes
Timer queue initialization (_MX_init_tq)	3 bytes	3 bytes

α: Stack size used by timer interrupt prohibit/enable routine

[MEMO]

Phase-out/Discontinued

NEC

Facsimile Message

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

From:

Name

Company

Tel. FAX

Address

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑


Phase-out/Discontinued