CUSTOMER NOTIFICATION

ZBB-CD-07-0054

January 31, 2007

Koji Nishibayashi, Group Manager
Development Tool Group
Multipurpose Microcomputer Systems Division
4th Systems Operations Unit
NEC Electronics Corporation

CP(K), O

78K0R Microcontroller Real-Time OS

RX78K0R Ver. 4.00

Development in Assembly Language

Be sure to read this document before using the product.

CONTENTS

This document explains methods for coding the 78K0R microcontroller real-time OS RX78K0R in assembly language.

1. CAUTIONS

1.1 Difference in Symbols Between C and Assembly Language

The symbols handled in the RX78K0R differ depending on whether the coding language is C or assembly language. Symbols used in assembly language code prefix an underscore to the corresponding C symbols.

An example is shown below.

```
<C>
  wai_flg

<Assembly language>
 _wai_flg
```

1.2 Saving Memory Area When Unused Service Calls Are Included

When the header file *kernel.inc*, provided by the RX78K0R, is included, all the service calls prepared by the real-time OS are linked by default.

When not linking the unused service calls to reduce memory consumption, the user must edit the contents of the header file inc78k0r\function.inc in the folder where the RX78K0R is installed.

Service call declarations are listed in *function.inc*, as shown below.

If there are unnecessary service calls, delete the line or comment it out.

```
 (Example)
<Initial state>                          <Example of edition when ref_tsk is unnecessary>
extrn       _chg_pri                     extrn       _chg_pri
extrn       _ichg_pri                    extrn       _ichg_pri
extrn       _ref_tsk                     ;extrn      _ref_tsk
 :
```

To include function.inc in source code, include kernel.inc, which includes function.inc internally.

1.3 Coding Application Code Only in Assembly Language

When coding the application only in assembly language files, be sure to define symbols as shown below and link them to the files. These definitions are provided in sample programs. Use smp78k0r\QB-78K0RKX3-asm\appli\src\saddr.asm, located in the folder where the RX78K0R is installed.

```
        public  _@RTARG0
        public  _@RTARG2
        public  _@RTARG4
        public  _@RTARG6
        public  _@SEGAX
        public  _@SEGDE
        public  _@KREG00
        public  _@KREG02
        public  _@KREG04
        public  _@KREG06
        public  _@KREG08
        public  _@KREG10
        public  _@KREG12
        public  _@KREG14

@@CCUSESEG      DSEG    SADDRP
_@RTARG0:       DS      (2)
_@RTARG2:       DS      (2)
_@RTARG4:       DS      (2)
_@RTARG6:       DS      (2)
_@SEGAX:        DS      (2)
_@SEGDE:        DS      (2)
_@KREG00:       DS      (2)
_@KREG02:       DS      (2)
_@KREG04:       DS      (2)
_@KREG06:       DS      (2)
_@KREG08:       DS      (2)
_@KREG10:       DS      (2)
_@KREG12:       DS      (2)
_@KREG14:       DS      (2)
END
```

The RA78K0R returns the following error if the application code is written only in assembly language files without defining the above symbols.

RA78K0R error E3405: Undefined symbol '_@RTARG00 in file 'XXXX'

2. CODING OF PROGRAM PROCESSING ROUTINES

2.1 Boot Processing

This is system initialization processing that performs initialization of any hardware or memory area, and then calls RX78K0R initialization processing __urx_start.

__urx_start has no arguments. The system stack is used for __urx_start processing. Setting of specific stack for __urx_start is unnecessary because __urx_start switches the stack used before calling to the system stack.

An example of boot processing is shown below, assuming that the uPD78F1165_A0 is used.

Basically, the register bank number set before calling __urx_start cannot be changed, except for cases where it is changed in an interrupt handler not controlled by the kernel, as long as the register bank before and after interrupt servicing is guaranteed.

```
NAME    @cstart

;---------------------------------------------------------------------
; Declaration of symbol
;---------------------------------------------------------------------

PUBLIC  _@cstart,_@cend

;---------------------------------------------------------------------
; External declaration of symbol for stack area
;
; _@STBEG has value of the end address +1 of compiler's stack area.
; _@STBEG is created by linker with -S option.
; Accordingly, specify the -S option when linking.
;
; _@MAA has value of the mirror area.
;---------------------------------------------------------------------
            EXTRN   __urx_start,_@STBEG,_hdwinit


@@VECT00        CSEG    AT      0
            DW      _@cstart

@@LCODE         CSEG    BASE

_@cstart:
;---------------------------------------------------------------------
; Setting of register bank (RB0, for example)
;---------------------------------------------------------------------
            SEL     RB0                         ; Sets register bank

;---------------------------------------------------------------------
; Setting of stack pointer
; Unnecessary if stack is not used by boot processing.
; The _@STBEG symbol here is generated by linker option -S.
;---------------------------------------------------------------------
            MOVW    SP,#LOWW _@STBEG        ; Sets the stack start address to the SP
            CALL    !!_hdwinit              ; Hardware initialization processing (user's option)
```

```
;---------------------------------------------------------------------------
; Clears saddr area
;---------------------------------------------------------------------------
        MOV     B,#0FEDFH-0FE20H+1
        CLRW    AX
LSADR1:
        DEC     B
        DEC     B
        MOVW    0FE20H[B],AX
        BNZ     $LSADR1

;---------------------------------------------------------------------------
; Clears RAM area
;---------------------------------------------------------------------------
        MOV     ES,#0FH
        MOVW    BC,#0FE20H-0D700H
        CLRW    AX
LSADR2:
        DECW    BC
        DECW    BC
        MOVW    0D700H[BC],AX
        CMPW    AX,BC
        BNZ     $LSADR2

;---------------------------------------------------------------------------
; Calling of RX78K0R initialization processing
;---------------------------------------------------------------------------
        BR      !!__urx_start
;---------------------------------------------------------------------------
; End loop
;---------------------------------------------------------------------------
        BR      $$
;
_@cend:
END
```

Insertion of main processing of hardware initialization function _hdwinit depends on the user. A reference program is shown below.

```
        PUBLIC  _hdwinit

        CSEG
_hdwinit:
        ; timer INTTM00
        MOV     0FFFA4H, #09H           ; CKC     = 0x09
        ONEB    !0F0H                   ; PER0    = 0x01
        MOVW    0FFF18H, #0FA0H         ; TDR00 = 4 * 1000

        ; INTTM00 priORity 2
        ONEW    AX                      ; TS0     = 0x0001
        MOVW    !01B2H, AX              ;
        MOV     A, 0FFFEAH              ; PR0_TM &= ~0x10
        AND     A, #0EFH        ;
        MOV     0FFFEAH, A              ;
        MOV     A, 0FFFEEH              ;
        OR      A, #010H                ; PR1_TM |=   0x10
        MOV     0FFFEEH, A              ;

        ; INTTM00 mask
        MOV     A, 0FFFE6H              ; MKR_TM0 &= ~0x10
        AND     A, #0EFH        ;
        MOV     0FFFE6H, A              ;

        ; inthANDler INTP0
        ; INTP0 priORity 3
        MOV     A, 0FFFE8H              ; PR0_INT |= 0x04
        OR      A, #04H                 ;
        MOV     0FFFE8H, A              ;
        MOV     A, 0FFFECH              ; PR1_INT |= 0x04
        OR      A, #04H                 ;
        MOV     0FFFECH, A              ;

        MOV     A, 0FFFE4H              ; MKR_INT0 &= ~0x04
        AND     A, #0FBH        ;
        MOV     0FFFE4H, A              ;

        RET

    END
```

2.2 Initialization Routine

This is the routine used for software initialization, which is called after kernel initialization by __urx_start is completed.

The symbol name at the start address is fixed to "_init_handler". This routine does not have arguments. The system stack is used during processing. This processing ends with the RET instruction. The scheduler starts up after the initialization routine ends, and the task with the highest priority is activated.

```
    $ INCLUDE (kernel.inc)          ; Includes header files in which kernel functions is defined

        PUBLIC  _init_handler

        CSEG
_init_handler:
                                    ; Main processing

        RET                         ; Ends initialization routine
        END
```

2.3 Task

A task is the minimum processing unit. A task's extended information *exinf* (4 bytes) is passed as an argument. The task stack unique to each task is used during task processing. The processing ends with the service call *ext_tsk*.

```
$ INCLUDE (kernel.inc)          ; Includes header files in which kernel functions is defined
$ INCLUDE (kernel_id.inc)       ; Includes header files in which kernel information is
defined

        PUBLIC  _task1

        CSEG
_task1:                         ; Task function (_task1) start address
        PUSH    BC              ; Saves task argument (upper 2 bytes of exinf) to the stack
        PUSH    AX              ; Saves task argument (lower 2 bytes of exinf) to the stack
                                ; The above saving unnecessary if exinf is not used.

        MOVW    AX, #ID_TASK1   ; Main processing. Here, a task with task ID "ID_TASK1"
        CALL    !!_act_tsk      ; is activated by the service call act_tsk.

        BR      !!_ext_tsk      ; Calls task end processing. Unnecessary to restore the stack.
        END
```

2.4 Idle Routine

This is the routine that is activated when there are no tasks in Ready state.

The symbol name at the start address is fixed to "_idle_handler". This routine does not have arguments. The system stack is used during processing. This processing ends with the RET instruction, but execution returns to the top of _idle_handler and continues to loop until a task in Ready state appears.

```
        PUBLIC  _idle_handler

        CSEG
_idle_handler:
        HALT                    ; Main processing. Here, processing enters power-save
                                ; mode due to HALT instruction

        RET                     ; Ends idle routine
        END
```

2.5 Interrupt Handler

This is the routine that is activated when an interrupt occurs. This routine does not have arguments.

An interrupt handler is activated when an interrupt occurs by registering the start address of the interrupt handler to the vector table.

The __kernel_int_entry function is called at the beginning of an interrupt handler. The following is performed during this processing.

- Reporting interrupt start to OS
  Reports to the OS that an interrupt has been started.
- Saving register contents
  Saves the contents of general-purpose registers (AX, BC, DE and HL), and the ES and CS registers to the stack of the processing program in which the interrupt occurred. (The contents of the PSW and PC have already been saved by the device.)
- Switching to system stack
  Switches the stack of the task in which the interrupt occurred to the system stack.

The _ret_int function is called at the end of an interrupt handler. The following is performed during this processing.

- Reporting interrupt end to OS
  Reports to the OS that the interrupt routine has ended.
- Calling scheduler
  Calls the scheduler. If no task switching takes place as a result of service call execution by the interrupt handler, the execution exits the scheduler and returns to the task in which the interrupt occurred.
  If task switching does take place, the execution does not return to the task immediately. When it returns depends on the subsequent schedule of the task.
- Restoring the register contents
  Restores the contents of general-purpose registers (AX, BC, DE and HL), and the ES and CS registers to the stack of the processing program in which the interrupt occurred. (The contents of the PSW and PC have already been saved by the device.)
- Switching to stack of task in which interrupt occurred
  Switches the system stack to the stack of the task in which the interrupt occurred.

The user must save and restore the contents of _@RTARGxx, _@SEGAX and _@SEGDE in the *saddr* area before and after interrupt handler main processing.

Saving of the *saddr* area contents is unnecessary if no C functions or service calls are called in interrupt handler processing.

```
$ INCLUDE (kernel.inc)
$ INCLUDE (kernel_id.inc)

        PUBLIC  _inthdr1
        EXTRN   _@RTARG0
        EXTRN   _@RTARG2
        EXTRN   _@RTARG4
        EXTRN   _@RTARG6
        EXTRN   _@SEGAX
        EXTRN   _@SEGDE

        CSEG
_inthdr1:
        CALL    !!__kernel_int_entry        ; Starts interrupt handler. Registers saved internally.
        MOVW    AX, _@RTARG0                ; Saves saddr area
        PUSH    AX
        MOVW    AX, _@RTARG2
        PUSH    AX
        MOVW    AX, _@RTARG4
        PUSH    AX
        MOVW    AX, _@RTARG6
        PUSH    AX
        MOVW    AX, _@SEGAX
        PUSH    AX
        MOVW    AX, _@SEGDE
        PUSH    AX
        ; Main processing
        POP     AX
        MOVW    _@SEGDE, AX                 ; Restores saddr area
        POP     AX
        MOVW    _@SEGAX, AX
        POP     AX
        MOVW    _@RTARG6, AX
        POP     AX
        MOVW    _@RTARG4, AX
        POP     AX
        MOVW    _@RTARG2, AX
        POP     AX
        MOVW    _@RTARG0, AX
        BR      !!_ret_int                  ; Ends interrupt handler. Stack returned internally.

INTP0   CSEG    AT      08h                 ; Registers the interrupt handler to vector table
        DW      _inthdr1
END
```

2.6 Timer Handler

This is the routine that is used as the kernel base clock. The code is almost the same as that of interrupt handlers. The only difference is that timer handler main processing _Timer_Handler is called.

```
$ INCLUDE (kernel.inc)
$ INCLUDE (kernel_id.inc)

        PUBLIC  _tmrhdr
        EXTRN   _@RTARG0
        EXTRN   _@RTARG2
        EXTRN   _@RTARG4
        EXTRN   _@RTARG6
        EXTRN   _@SEGAX
        EXTRN   _@SEGDE

        CSEG
_tmrhdr:
        CALL    !!__kernel_int_entry        ; Starts timer handler. Registers saved internally
        MOVW    AX, _@RTARG0                ; Saves saddr area
        PUSH    AX
        MOVW    AX, _@RTARG2
        PUSH    AX
        MOVW    AX, _@RTARG4
        PUSH    AX
        MOVW    AX, _@RTARG6
        PUSH    AX
        MOVW    AX, _@SEGAX
        PUSH    AX
        MOVW    AX, _@SEGDE
        PUSH    AX
        CALL    !!_Timer_Handler            ; Calls timer handler main processing
        POP     AX
        MOVW    _@SEGDE, AX                 ; Restores saddr area
        POP     AX
        MOVW    _@SEGAX, AX
        POP     AX
        MOVW    _@RTARG6, AX
        POP     AX
        MOVW    _@RTARG4, AX
        POP     AX
        MOVW    _@RTARG2, AX
        POP     AX
        MOVW    _@RTARG0, AX
        BR      !!_ret_int                  ; Ends timer handler. Stack returned internally.

INTTM00 CSEG    AT      2ch                 ; Registers timer handler to vector table
        DW      _tmrhdr
END
```

2.7 Cyclic Handler

This is the routine that is called from timer handlers and activated cyclically. This routine does not have arguments. The system stack is used during processing. This processing ends with the RET instruction.

```
$ INCLUDE(kernel.inc)              ; Includes header files in which kernel functions is defined
$ INCLUDE (kernel_id.inc)          ; Includes header files in which kernel information is defined

        PUBLIC  _cychdr1

        CSEG
_cychdr1:
        ; Main processing

        RET                        ; Ends interrupt handler
        END
```

3. SERVICE CALLS

3.1 Information Saved/Restored by Service Calls and Stack Size Used

The PC, PSW, and HL register values are saved at the beginning of service calls, and restored when the service call ends.

These values are saved to the stack of the processing program that called the service call (task stack or system stack) immediately after the service call processing is started.

The system stack is mainly used during service call processing. However, the stack of the processing program that called the service call is used for a short term at the beginning and end of service call processing.

The following shows the maximum stack size used when a service call is called (unit: bytes).

In the table below,

“Stack used by arguments” refers to the stack used by the user program before the service call is called.

“Stack used by OS” refers to the stack used by OS processing.

“Stack of program that called service call” refers to the stack used when the service call is called. It is called the “task stack” if the service call is called from a task, or the “system stack” if the service call is called from an interrupt handler, timer handler, or cyclic handler. Calling of service calls from any other processing is prohibited.

In the case of the system stack, the stack size used is the total of the “stack of program that called service call” and the “system stack”.

Type	Service Call	Stack Used by Arguments	Stack Used by OS	
		Stack of Program That Called Service Call		System Stack
Task management functions	act_tsk	0	10	4
	iact_tsk	0	10	4
	can_act	0	8	4
	sta_tsk	4	8	8
	ista_tsk	4	8	8
	ext_tsk	0	8	4
	ter_tsk	0	8	4
	chg_pri	2	8	4
	ichg_pri	2	8	4
	ref_tsk	4	8	4
Task dependent synchronization functions	slp_tsk	0	8	4
	tslp_tsk	0	8	4
	wup_tsk	0	8	4
	iwup_tsk	0	8	4
	can_wup	0	8	4
	ican_wup	0	8	4
	rel_wai	0	8	4
	irel_wai	0	8	4
	sus_tsk	0	8	4

Type	Service Call	Stack Used by Arguments	Stack Used by OS	
		Stack of Program That Called Service Call		System Stack
Task dependent synchronization functions	isus_tsk	0	8	4
	rsm_tsk	0	8	4
	irsm_tsk	0	8	4
	frsm_tsk	0	8	4
	ifrsm_tsk	0	8	4
	dly_tsk	0	8	4
Semaphores	sig_sem	0	8	4
	isig_sem	0	8	4
	wai_sem	0	8	4
	pol_sem	0	8	4
	twai_sem	4	8	4
	ref_sem	4	8	4
Event flags	set_flg	2	8	4
	iset_flg	2	8	4
	clr_flg	2	8	4
	wai_flg	8	8	6
	pol_flg	8	8	6
	twai_flg	12	8	6
	ref_flg	4	8	4
Mailboxes	snd_mbx	4	8	8
	rcv_mbx	4	8	6
	prcv_mbx	4	8	6
	trcv_mbx	8	8	6
	ref_mbx	4	8	4
Memory pool management functions	get_mpf	4	8	6
	pget_mpf	4	8	6
	tget_mpf	8	8	6
	rel_mpf	4	8	4
	ref_mpf	4	8	4
Time management functions	sta_cyc	0	8	4
	stp_cyc	0	8	4
	ref_cyc	4	8	4
Other	rot_rdq	0	8	4
	irot_rdq	0	8	4
	get_tid	0	8	4
	iget_tid	0	8	4
	loc_cpu	0	8	4
	iloc_cpu	0	8	4
	unl_cpu	0	8	4
	iunl_cpu	0	8	4
	dis_dsp	0	8	4
	ena_dsp	0	8	4
	sns_ctx	0	8	4
	sns_loc	0	8	4
	sns_dsp	0	8	4
	sns_dpn	0	8	4
	ret_int	0	0	0
	ref_ver	0	8	4

The following lists special functions that are not classified as service calls.

The system stack is used by __urx_start and _TImer_Handler.

In the case of __kernel_int_entry, the stack is switched to the system stack after the register contents are saved to the stack of the program that called the function.

Type	Service Call	Stack Used by Arguments	Stack Used by OS	
		Stack of Program That Called Service Call		System Stack
Functions	__urx_start	0	-	4
	__kernel_int_entry	0	14	2
	_Timer_Handler	0	-	6

3.2 Calling Methods of Service Call

Arguments of a service call are stored in the AX register if the first argument size is 1 or 2 bytes, or in the AX register (upper) and the BC register (lower) if it is 3 or 4 bytes.

The second and subsequent arguments are stored in the stack.

Values returned from a service call are stored in the C register for sizes of 1 byte, or in the BC register for sizes of 2 bytes.

Example: The following shows an example of calling the service call *wai_flg*, coded in assembly language.

The specifications of arguments passed to *wai_flg* are as follows.

Argument	Name ← Value	Size	Description
First argument	flgid ← ID_FLG1	1 byte	Event flag ID
Second argument	waiptn ← WAIT_FLGPTN	2 bytes	Requested bit pattern
Third argument	wfmode ← TWF_ORW	1 byte	Specification of request condition
Fourth argument	p_flgptn ← ES (upper 2 bytes) + p_flgptn_lo (lower 2 bytes)	4 bytes	Address at which the bit pattern when the condition is established is stored

The following shows an example of calling *wai_flg*.

```
MOV     A, ES
MOV     C, A                ; Sets ES value (upper 2 bytes of 4th argument) to C register
MOVW    DE, #p_flgptn_lo    ; Sets 4th argument value (p_flgptn_lo) to DE register
PUSH    BC                  ; Saves upper 2 bytes of 4th argument to stack
                            ; B register value can be undefined
PUSH    DE                  ; Saves lower 2 bytes of 4th argument to stack
MOVW    AX, #TWF_ORW        ; Sets 3rd argument value (TWF_ORW) to AX register
PUSH    AX                  ; Saves 2 bytes of 3rd argument to stack
MOVW    AX, #WAIT_FLGPTN    ; Sets 2nd argument value (WAIT_FLGPTN) to AX register
PUSH    AX                  ; Saves 2 bytes of 2nd argument to stack
MOVW    AX, #ID_FLG1        ; Sets 1st argument value (ID_FLG1) to AX register
CALL    !!_wai_flg          ; Calls service call wai_flg
ADDW    SP, #08H            ; Restores the total of 8 stack bytes used for arguments.
                            ; Restores no argument values because none are used in future.
```

The following shows the *wai_flg* type definition in C.

```
ER          wai_flg(ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN * p_flgptn);
```

If *wai_flg* is written in C, the code is as follows.

```
FLGPTN   p_flgptn;
wai_flg(ID_FLG1, WAIT_FLGPTN, TWF_ORW, &p_flgptn);
```

Reference: Data types and their sizes in C

The macro names of data types used as service call arguments or return values, actual types, and their data sizes in C code are as follows.

Macro	Type in C	Size (Bytes)
ID	unsigned char	1
BOOL	signed char	1
MODE	unsigned char	1
PRI	signed char	1
ER	signed short int	2
ER_UINT	signed int	2
FLGPTN	unsigned int	2
STAT	unsigned short int	2
UINT	unsigned int	2
RELTIM	unsigned long int	4
TMO	signed long int	4
VP_INT	signed long int	4
*VP	void __far	4
–	Pointer type	4

3.3 Service Call Calling Examples

This section shows examples of calling service calls in assembly language for the corresponding C syntax.

This section presumes that the following conditions are satisfied. If the actual coding conditions differ from these presumptions, modify the example code according to the actual conditions.

- The values of general-purpose registers AX, BC and DE are not held before and after service call processing. These values must be saved and restored if they are used after service call processing.
- Return values from service calls to the BC register are not manipulated.
- Values in the ES register are always read in accessing the address data. It is unnecessary to read values from the ES register if the ES register values are known.

Type	Service Call	Syntax in C	Example of Calling in Assembly Language
Task management functions	act_tsk	ER act_tsk(ID tskid);	MOVW AX, #tskid CALL !!_act_tsk
	iact_tsk	ER iact_tsk(ID tskid);	MOVW AX, #tskid CALL !!_iact_tsk
	can_act	ER_UINT can_act(ID tskid);	MOVW AX, #tskid CALL !!_can_act

	sta_tsk	ER
		sta_tsk(ID tskid, VP_INT stacd);
			MOVW AX, #stacd_lo
			PUSH AX
			MOVW AX, #stacd_hi
			PUSH AX
			MOVW AX, #tskid
			CALL !!_sta_tsk
			addw sp, #04H

	ista_tsk	ER
		ista_tsk(ID tskid, VP_INT stacd);
			MOVW AX, #stacd_hi
			PUSH AX
			MOVW AX, #stacd_lo
			PUSH AX
			MOVW AX, #tskid
			CALL !!_ista_tsk
			addw sp, #04H

	ext_tsk	void ext_tsk(void);	BR !!_ext_tsk
	ter_tsk	ER ter_tsk(ID tskid);	MOVW AX, #tskid CALL !!_ter_tsk

	chg_pri	ER
		chg_pri(ID tskid, PRI tskpri);
			MOVW AX, #tskpri
			PUSH AX
			MOVW AX, #tskid
			CALL !!_chg_pri
			POP AX

	ichg_pri	ER
		ichg_pri(ID tskid, PRI tskpri);
			MOVW AX, #tskpri
			PUSH AX
			MOVW AX, #tskid
			CALL !!_ichg_pri
			POP AX

	ref_tsk	ER
		ref_tsk(ID tskid, T_RTSK *pk_rtsk);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rtsk_lo
			PUSH BC
			PUSH DE
			MOVW AX, #tskid
			CALL !!_ref_tsk
			addw sp, #04H

Type	Service Call	Syntax in C	Example of Calling in Assembly Language
Task dependent synchronization functions	slp_tsk	ER slp_tsk(void);	CALL !!_slp_tsk

	tslp_tsk	ER
		tslp_tsk(TMO tmout);
			MOVW AX, #tmout_lo
			MOVW BC, #tmout_hi
			CALL !!_tslp_tsk

	wup_tsk	ER wup_tsk(ID tskid);	MOVW AX, #tskid CALL !!_wup_tsk
	iwup_tsk	ER iwup_tsk(ID tskid);	MOVW AX, #tskid CALL !!_iwup_tsk
	can_wup	ER_UINT can_wup(ID tskid);	MOVW AX, #tskid CALL !!_can_wup
	ican_wup	ER_UINT ican_wup(ID tskid);	MOVW AX, #tskid CALL !!_ican_wup
	rel_wai	ER rel_wai(ID tskid);	MOVW AX, #tskid CALL !!_rel_wai
	irel_wai	ER irel_wai(ID tskid);	MOVW AX, #tskid CALL !!_irel_wai
	sus_tsk	ER sus_tsk(ID tskid);	MOVW AX, #tskid CALL !!_sus_tsk
	isus_tsk	ER isus_tsk(ID tskid);	MOVW AX, #tskid CALL !!_isus_tsk
	rsm_tsk	ER rsm_tsk(ID tskid);	MOVW AX, #tskid CALL !!_rsm_tsk
	irsm_tsk	ER irsm_tsk(ID tskid);	MOVW AX, #tskid CALL !!_irsm_tsk
	frsm_tsk	ER frsm_tsk(ID tskid);	MOVW AX, #tskid CALL !!_frsm_tsk
	ifrsm_tsk	ER ifrsm_tsk(ID tskid);	MOVW AX, #tskid CALL !!_ifrsm_tsk

	dly_tsk	ER
		dly_tsk(RELTIM dlytim);
			MOVW AX, #dlytim_lo
			MOVW BC, #dlytim_hi
			CALL !!_dly_tsk

Synchronization and communication functions (semaphores)	sig_sem	ER sig_sem(ID semid);	MOVW AX, #semid CALL !!_sig_sem
	isig_sem	ER isig_sem(ID semid);	MOVW AX, #semid CALL !!_isig_sem
	wai_sem	ER wai_sem(ID semid);	MOVW AX, #semid CALL !!_wai_sem
	pol_sem	ER pol_sem(ID semid);	MOVW AX, #semid CALL !!_pol_sem

	twai_sem	ER
		twai_sem(ID semid, TMO tmout);
			MOVW AX, #tmout_hi
			PUSH AX
			MOVW AX, #tmout_lo
			PUSH AX
			MOVW AX, #semid
			CALL !!_twai_sem
			addw sp, #04H

	ref_sem	ER
		ref_sem(ID semid, T_RSEM
		*pk_rsem);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rsem_lo
			PUSH BC
			PUSH DE
			MOVW AX, #semid
			CALL !!_ref_sem
			addw sp, #04H

Type	Service Call	Syntax in C	Example of Calling in Assembly Language

Synchronization and communication functions (semaphores)	set_flg	ER
		set_flg(ID flgid, FLGPTN setptn);
			MOVW AX, #setptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_set_flg
			POP AX

	iset_flg	ER
		iset_flg(ID flgid,FLGPTN setptn);
			MOVW AX, #setptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_iset_flg
			POP AX

	clr_flg	ER
		clr_flg(ID flgid, FLGPTN clrptn);
			MOVW AX, #clrptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_clr_flg
			POP AX

	wai_flg	ER
		wai_flg(ID flgid, FLGPTN waiptn,
		MODE wfmode,
		FLGPTN * p_flgptn);
			MOV A, ES
			MOV C, A
			MOVW DE, #p_flgptn_lo
			PUSH BC
			PUSH DE
			MOVW AX, #wfmode
			PUSH AX
			MOVW AX, #waiptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_wai_flg
			addw sp, #08H

	pol_flg	ER
		pol_flg(ID flgid, FLGPTN waiptn,
		MODE wfmode,
		FLGPTN * p_flgptn);
			MOV A, ES
			MOV C, A
			MOVW DE, #p_flgptn_lo
			PUSH BC
			PUSH DE
			MOVW AX, #wfmode
			PUSH AX
			MOVW AX, #waiptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_pol_flg
			addw sp, #08H

	twai_flg	ER
		twai_flg(ID flgid, FLGPTN waiptn,
		MODE wfmode,
		FLGPTN * p_flgptn
		TMO tmout);
			MOVW AX, #tmout_hi
			PUSH AX
			MOVW AX, #tmout_lo
			PUSH AX
			MOV A, ES
			MOV C, A
			MOVW DE, #p_flgptn_lo
			PUSH BC
			PUSH DE
			MOVW AX, #wfmode
			PUSH AX
			MOVW AX, #waiptn
			PUSH AX
			MOVW AX, #flgid
			CALL !!_twai_flg
			addw sp, #0CH

	ref_flg	ER
		ref_flg(ID flgid, T_RFLG *pk_rflg);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rflg_lo
			PUSH BC
			PUSH DE
			MOVW AX, #flgid
			CALL !!_ref_flg
			addw sp, #04H

Type	Service Call	Syntax in C	Example of Calling in Assembly Language

Synchronization and communication functions (mailboxes)	snd_mbx	ER
		snd_mbx(ID mbxid, T_MSG * pk_msg);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_msg_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mbxid
			CALL !!_snd_mbx
			addw sp, #04H

	rcv_mbx	ER
		rcv_mbx(ID mbxid, T_MSG **
		ppk_msg);
			MOV A, ES
			MOV C, A
			MOVW DE, #ppk_msg_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mbxid
			CALL !!_rcv_mbx
			addw sp, #04H

	prcv_mbx	ER
		prcv_mbx(ID mbxid, T_MSG **
		ppk_msg);
			MOV A, ES
			MOV C, A
			MOVW DE, #ppk_msg_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mbxid
			CALL !!_prcv_mbx
			addw sp, #04H

	trcv_mbx	ER
		trcv_mbx(ID mbxid, T_MSG **
		ppk_msg
		TMO tmout);
			MOVW AX, #tmout_hi
			PUSH AX
			MOVW AX, #tmout_lo
			PUSH AX
			MOV A, ES
			MOV C, A
			MOVW DE, #ppk_msg_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mbxid
			CALL !!_trcv_mbx
			addw sp, #08H

	ref_mbx	ER
		ref_mbx(ID mbxid, T_RMBX
		*pk_rmbx);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rmbx_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mbxid
			CALL !!_ref_mbx
			addw sp, #04H

Type	Service Call	Syntax in C	Example of Calling in Assembly Language

Memory pool management functions	get_mpf	ER
		get_mpf(ID mpfid, VP *p_blk);
			MOV A, ES
			MOV C, A
			MOVW DE, #p_blk_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mpfid
			CALL !!_get_mpf
			addw sp, #04H

	pget_mpf	ER
		pget_mpf(ID mpfid, VP *p_blk);
			MOV A, ES
			MOV C, A
			MOVW DE, #p_blk_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mpfid
			CALL !!_pget_mpf
			addw sp, #04H

	tget_mpf	ER
		tget_mpf(ID mpfid, VP *p_blk, TMO tmout);
			MOVW AX, #tmout_hi
			PUSH AX
			MOVW AX, #tmout_lo
			PUSH AX
			MOV A, ES
			MOV C, A
			MOVW DE, #p_blk_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mpfid
			CALL !!_tget_mpf
			addw sp, #08H

	rel_mpf	ER
		rel_mpf(ID mpfid, VP blk);
			MOV A, ES
			MOV C, A
			MOVW DE, #blk_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mpfid
			CALL !!_rel_mpf
			addw sp, #04H

	ref_mpf	ER
		ref_mpf(ID mpfid, T_RMPF *pk_rmpf);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rmpf_lo
			PUSH BC
			PUSH DE
			MOVW AX, #mpfid
			CALL !!_ref_mpf
			addw sp, #04H

Type	Service Call	Syntax in C	Example of Calling in Assembly Language
Time management functions	sta_cyc	ER sta_cyc(ID cycid);	MOVW AX, #cycid CALL !!_sta_cyc
	stp_cyc	ER stp_cyc(ID cycid);	MOVW AX, #cycid CALL !!_stp_cyc

	ref_cyc	ER
		ref_cyc(ID cycid, T_RCYC *pk_rcyc);
			MOV A, ES
			MOV C, A
			MOVW DE, #pk_rcyc_lo
			PUSH BC
			PUSH DE
			MOVW AX, #cycid
			CALL !!_ref_cyc
			addw sp, #04H

Other	rot_rdq	ER rot_rdq(PRI tskpri);	MOVW AX, #tskpri CALL !!_rot_rdq
	irot_rdq	ER irot_rdq(PRI tskpri);	MOVW AX, #tskpri CALL !!_irot_rdq

	get_tid	ER
		get_tid(ID *p_tskid);
			MOVW AX, #p_tskid_lo
			MOVW BC, #p_tskid_hi
			CALL !!_get_tid

	iget_tid	ER
		iget_tid(ID *p_tskid);
			MOVW AX, #p_tskid_lo
			MOVW BC, #p_tskid_hi
			CALL !!_iget_tid

	loc_cpu	ER loc_cpu(void);	CALL !!_loc_cpu
	iloc_cpu	ER iloc_cpu(void);	CALL !!_iloc_cpu
	unl_cpu	ER unl_cpu(void);	CALL !!_unl_cpu
	iunl_cpu	ER iunl_cpu(void);	CALL !!_iunl_cpu
	dis_dsp	ER dis_dsp(void);	CALL !!_dis_dsp
	ena_dsp	ER ena_dsp(void);	CALL !!_ena_dsp
	sns_ctx	BOOL sns_ctx(void);	CALL !!_sns_ctx
	sns_loc	BOOL sns_loc(void);	CALL !!_sns_loc
	sns_dsp	BOOL sns_dsp(void);	CALL !!_sns_dsp
	sns_dpn	BOOL sns_dpn(void);	CALL !!_sns_dpn
	ret_int	void ret_int(void);	BR !!_ret_int

	ref_ver	ER ref_ver(T_RVER *pk_rver);	MOV A, ES
			MOV C, A
			MOVW AX, #pk_rver_lo
			CALL !!_ref_ver