32170/32174 Group, 32171 Group, 32176 Group

Example of Startup Program

1. Overview

The following article shows sample of general-purpose startup program used into 32170/32174 group, 32171 group, 32176 group.

2. Introduction

The explanation of this issue is applied to the following condition:

• Microcomputer:	32170/32174 Group 32171 Group 32176 Group
• Operation frequency:	20 to 40 MHz

3. Summary of the startup program

For an embedded application program to be run on the target system, you must have a startup program to initialize the target system, in addition to the user program and an EIT (Exception, Interrupt, Trap) handler routine.
EIT handler is detailed in chapter 4, “The Outline of EIT”

3.1 Method for Creating a Startup Program

To run an embedded application on the target system, you must have a program known as the “startup program” which calls up the user program (main function). The startup program performs initialization and processing, items (1) through (12) below.

(1) EIT Vector Entry
(2) Set Protect ID
(3) ICU Vector Table
(4) Allocate memory for the SFR area
(5) EIT handler processing
(6) Allocate memory for the stack area
(7) Set microprocessor operation mode
(8) Set the stack pointer
(9) Set the data area sections (B, D)
(10) Set Base register
(11) Call the main function
(12) IE bit Operation Function

The need for these processing varies with the contents of the actual application program.
Additional processing may be required, or some of the above processing may not be necessary.
In case the C standard library is used, it may be required to incorporate initializing and terminating.
For details refer to the “CC32R User's Manual C Compiler”.

3.1.1 EIT Vector Entry
(Lines 10 through 63 in Chapter 5, “Sample Programming Code”)

EIT vector entry is placed at the head of an internal ROM area or an extended area, and allocates a branch instruction to the processing handler start address (Note) of each EIT phenomenon. (Note: It is not branch address)
For details about EIT phenomenon refer to chapter 4, “The Outline of EIT”.

3.1.2 Setting Protect ID
(Lines 65 through 67 in Chapter 5, “Sample Programming Code”)

As general-purpose serial programmer, writing and erasing tools of internal flash memory in emulators and so on are in used, it checks out ID inputted with the tool against ID in an internal flash memory. Unless it is inputted right ID, writing and erasing operation are not accepted. (In some tools, after erasing all areas, it becomes possible to use tool operation and to write in internal flash memory.)
If it is not necessary to protect internal flash memory, set all checking ID areas for protection of an internal flash memory (H'0000 0084 to H'0000 0093) up as H'FF. It is assumed that protect ID is not necessary in this sample program.

3.1.3 ICU Vector Table
(Lines 70 through 109 in Chapter 5, “Sample Programming Code”)

Set ICU Vector Table (in the case of 32170, it is the address from H'0000 0094 to H'0000 010F) up as the start address of the interrupt handler of each internal peripheral I/O. As an interrupt request is accepted, set address of low 16 bits for ICU vector table that corresponds to sources of acception for it up as an Interrupt Vector Register (IVECT). As for EIT handler, the address of an ICU vector table is acquired by reading out the contents of this IVECT register under LDH command.

3.1.4 Allocating memory for the SFR area
(Lines 111 through 116 in Chapter 5, “Sample Programming Code”)

The register of internal peripheral I/O is allocated from address H'0080 0000. 16K bytes of SFR (Special Function Register) area is allocated here.

3.1.5 EIT handler processing
(Lines 120 through 197 in Chapter 5, “Sample Programming Code”)

If an EIT phenomenon occurs, the program that was being run till then is interrupted, and after running a hardware preprocessing (after-mentioned), it will branch to EIT handler processing.
In the case of this sample program, the EIT handler of external interrupt (EI) is realized.
For details refer to chapter 4.3, “Procedure for Processing of External Interrupts (EI) by EIT Handler”.
As system break interruption, exception and trap are occurred, handler processing is infinite loop processing.
In those cases, please describe the start address (label name) of the processing in the part where an EIT vector entry corresponds.

3.1.6 Allocating memory for the stack area (Lines 218 through 227 in Chapter 5, “Sample Programming Code”)

Allocate memory for the stack area required for the operation of the application. This generally is accomplished by declaring a stack area section with the pseudo-instruction. [.SECTION] and allocating memory for the stack area with the pseudo-instruction [.RES] in an assembly program.
The stack size required for the operation of the application is determined by considering the relationship between function calls and the stack size used by each function.
However, because it is difficult to find the exact stack size, a more practical method is adopted in which the necessary amount of stack is examined by actually running the application in the debug and evaluation processes. (For example, an ample stack area is allocated at the initial stage and then the necessary amount of stack is examined using a debugger.)
In addition, there is also the method of using a stack size calculation utility (stk32R). Stk32R is able to calculate stack size for operating program with adding “-stack” option and processing outputted stack quantity consumed indication file when compile. For details refer to the “CC32R User's Manual C Compiler”.
Because the stack is used in units of WORD (4 bytes), the stack area must be allocated in units of 4 bytes.
In this sample program 2048 bytes of stack area is secured. (It is necessary to secure a suitable stack area by the program in fact)

3.1.7 Setting microprocessor operation mode (Lines 234,235,297 through 299 in Chapter 5, “Sample Programming Code”)

Specify which stack to use and whether or not to enable interrupts for the target microprocessor by using the Processor Status Word (PSW) register.
The Processor Status Word (PSW) register indicates the status of the CPU, and consists of a normally used PSW field and a BPSW field used to save the PSW field when an EIT occurs.
In this sample program, settings are made to “disable interrupts” immediately after program execution, and to “use the interrupt stack” and “enable interrupts” immediately before calling the main function.

3.1.8 Setting the stack pointer (Lines 240 through 243 in Chapter 5, “Sample Programming Code”)

Set the most significant address of the allocated stack area in the stack pointer.
Because it has described that setting of interrupt stack is only used (stack mode bit (SM) is always “0”) on the assumption, stack for users is not set up.

3.1.9 Setting the data area sections (B, D)

Embedded applications require initializing the data area without initial values (section B) and the data area with initial values (section D).
Perform the following operations when linking the application and when executing the startup program:

(1) When linking the application

- Locate sections D and B in the RAM area. (Only allocate memory, without data output.).
- Locate the initial value data area for section D (ROM_D) in the ROM area.

(2) When initializing the startup program

- Initialize section B in the RAM area by clearing it to "0" (filling the area with 0s).
  (Lines 246 through 257 in Chapter 5, "Sample Programming Code")
- Initialize the data located in the ROM_D section of the ROM area by transferring it into section D in the RAM area.
  (Lines 260 through 273 in Chapter 5, "Sample Programming Code")

3.1.10 Setting base register
(Lines 276 through 292 in Chapter 5, "Sample Programming Code")

If the "-rel16" option is added when compiling, R11, R12, R13-based register relative indirect instruction is outputted for access to symbols for sections D and B.
Here, the program sets the R12 register and defines "_ _REL_BASE12" required for it. Because in this sample program, H'0080 8000 is defined for "_ _REL_BASE12", the register relative indirect instruction is output for the area ±32KB from H'0080 8000. And in case of adding "-access" option, an access control file is required in addition to setup the above-mentioned base register.
For details refer to the "CC32R User's Manual C Compiler".

3.1.11 Calling the main function
(Lines 301 in Chapter 5, "Sample Programming Code")

A subroutine call is set at the start address of the main function. For programs comprised not to return from the main function, a simple jump instruction can suffice.
For details, refer to the "CC32R User's Manual C Compiler".

3.1.12 IE bit operation function
(Lines 309 through 323 in Chapter 5, "Sample Programming Code")

- Interrupt enable function (EnInt)
  Interruption is enabled by setting the interrupt enable bit (IE) of PSW to "1".
- Interrupt disable function (DisInt)
  Interruption is disabled by clearing IE bit of PSW to "0".

4. The Outline of EIT

In process of CPU is running the usual program, if a certain phenomenon occurs, it may be necessary to interrupt the program and run another program. Such a phenomenon is generically called an EIT (Exception, Interrupt, and Trap) phenomenon.

4.1 Causes of EIT

There are following causes of EIT for the M32R/ECU.

(1) Exception

Exception occurs for errors or violations encountered during instruction execution, and include the following.

- Reserved Instruction Exception (RIE)
  This Exception occurs when an unimplemented instruction is detected.
- Address Exception (AE)
  This Exception occurs when an attempt is made to access non-aligned addresses in load or store instructions.

(2) Interrupt

Interrupts are generated for hardware causes or sources of the microcomputer, and include the following:

- Reset Interrupt (RI)
  It is generated when a reset signal is accepted.
  Reset interrupt has the highest priority.
- System Break Interrupt (SBI)
  This is an emergency interrupt generated when power-down condition is detected or when a faulty state is notified by an external watchdog timer, etc. This interrupt can only be used in cases where the interrupt handler does not return to the main routine.
- External Interrupt (EI)
  Interrupt request from each internal peripheral I/Os controlled by interrupt controller. The interrupt controller controls interrupt by 8 level priority order include interrupt disable.

(3) Trap

This refers to a software interrupt generated by executing the TRAP instruction in the program.

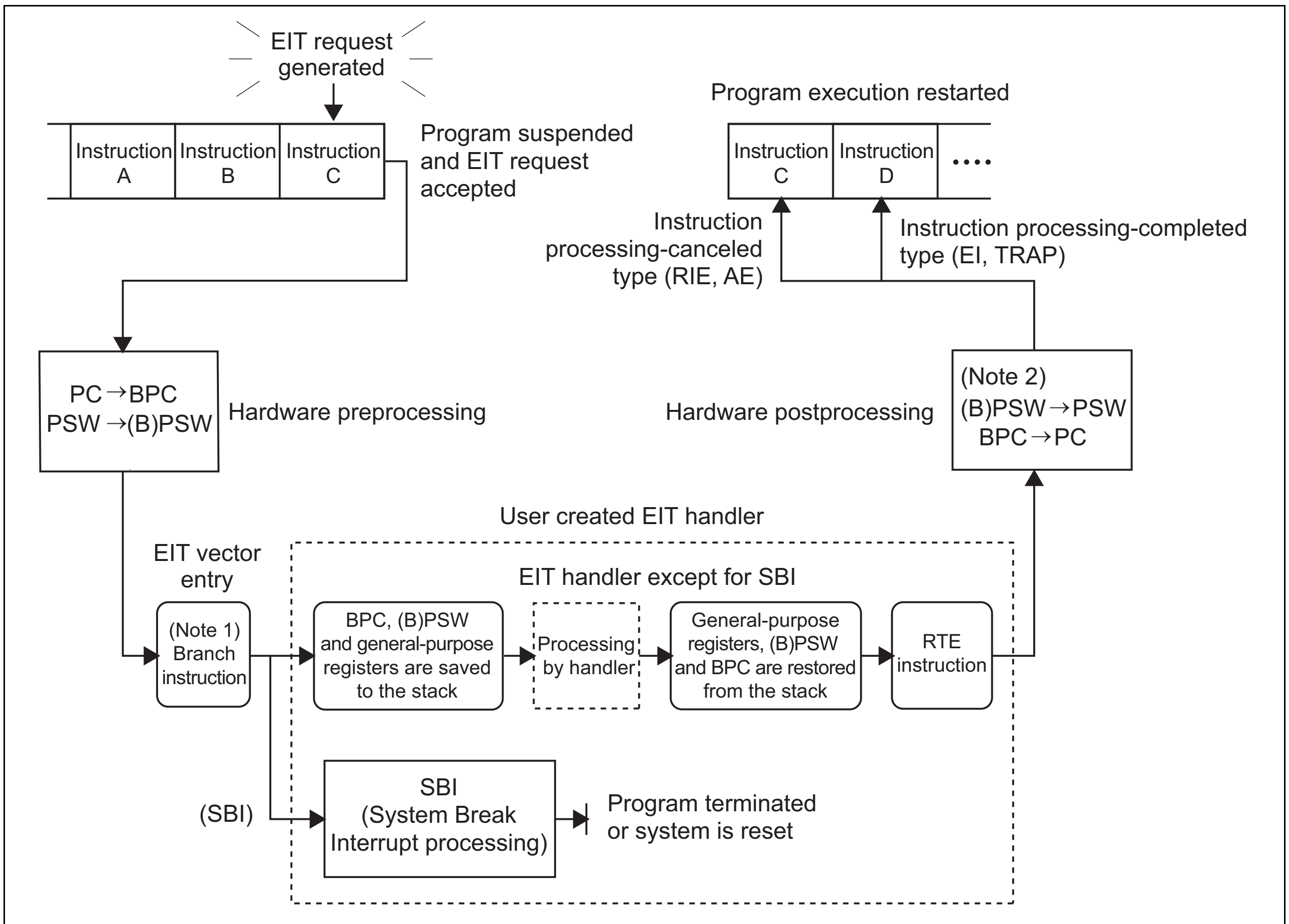
4.2 The Outline of EIT processing

As EIT phenomenon occurs, EIT processing takes place. In EIT processing there are areas which the hardware processes automatically or which the user routine program (EIT handler) processes.
EIT Processing Procedure except for Interrupt Reset is below.

(1) Processing in hardware when an EIT is accepted
- Register transfer: PC => BPC
- Register transfer: PSW => BPSW

(2) EIT handler
- Executes the instruction in the appropriate EIT vector entry for the cause of occurrence (branch instruction to the appropriate EIT handler, Lines 10 through 63 in Chapter 5, "Sample Programming Code")
- After executing the EIT handler, the processor exits from it with the RTE instruction. (except SBI)

(3) Processing in hardware after Command RTE
- Register transfer: BPSW => PSW
- Register transfer: BPC => PC

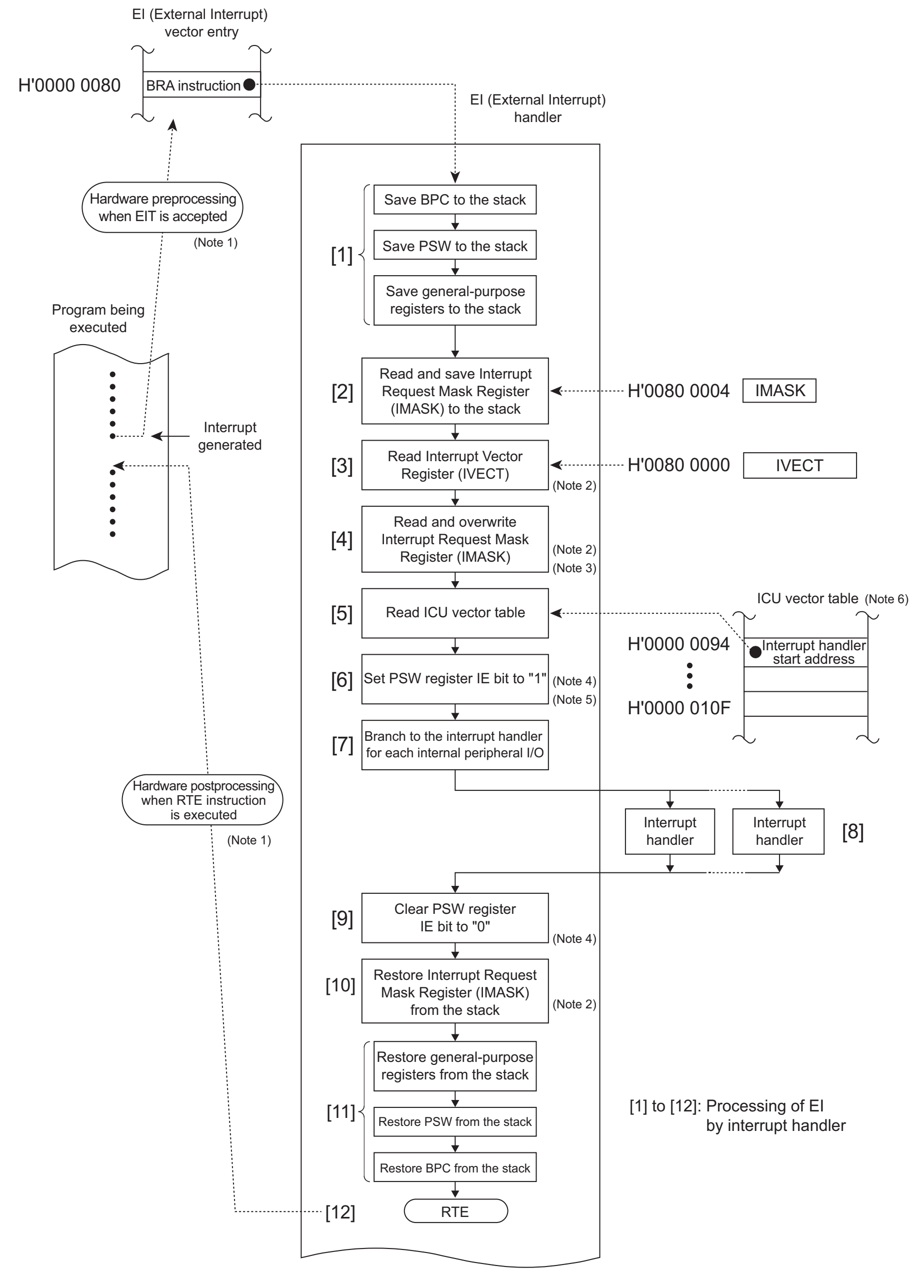

EIT request generated
Instruction A
Instruction B
Instruction C
Program suspended and EIT request accepted
Program execution restarted
Instruction C
Instruction D
....
Instruction processing-canceled type (RIE, AE)
Instruction processing-completed type (EI, TRAP)
PC→BPC
PSW→(B)PSW
Hardware preprocessing
Hardware postprocessing
(Note 2)
(B)PSW→PSW
BPC→PC
User created EIT handler
EIT vector entry
EIT handler except for SBI
(Note 1) Branch instruction
BPC, (B)PSW and general-purpose registers are saved to the stack
Processing by handler
General-purpose registers, (B)PSW and BPC are restored from the stack
RTE instruction
(SBI)
SBI (System Break Interrupt processing)
Program terminated or system is reset

Note 1: This is a branch instruction, and not the jump address.
Note 2: (B)PSW indicates the BPSW field for the PSW register.

Figure 4.2.1 Outline of EIT Processing Procedure

4.3 Procedure for Processing of External Interrupts (EI) by EIT Handler

The EIT handler processing procedure of external interrupts (EI) is shown below and also it is shown line 120 though 197 in chapter 5, “Sample Programming Code”.
It has optimized so that it can run at the maximum high speed considering a motion of a pipeline and so on.


EI (External Interrupt) vector entry
H'0000 0080
BRA instruction
EI (External Interrupt) handler
Hardware preprocessing when EIT is accepted
(Note 1)
Program being executed
Interrupt generated
Hardware postprocessing when RTE instruction is executed
(Note 1)
[1]
Save BPC to the stack
Save PSW to the stack
Save general-purpose registers to the stack
[2]
Read and save Interrupt Request Mask Register (IMASK) to the stack
H'0080 0004
IMASK
[3]
Read Interrupt Vector Register (IVECT)
(Note 2)
H'0080 0000
IVECT
[4]
Read and overwrite Interrupt Request Mask Register (IMASK)
(Note 2)
(Note 3)
[5]
Read ICU vector table
ICU vector table (Note 6)
H'0000 0094
Interrupt handler start address
H'0000 010F
[6]
Set PSW register IE bit to "1"
(Note 4)
(Note 5)
[7]
Branch to the interrupt handler for each internal peripheral I/O
Interrupt handler
Interrupt handler
[8]
[9]
Clear PSW register IE bit to "0"
(Note 4)
[10]
Restore Interrupt Request Mask Register (IMASK) from the stack
(Note 2)
[11]
Restore general-purpose registers from the stack
Restore PSW from the stack
Restore BPC from the stack
[12]
RTE
[1] to [12]: Processing of EI by interrupt handler

Note 1: For operations at EIT acceptance and return from EIT, also see Section 4.2, "The Outline of EIT Processing."
Note 2: Do not read the Interrupt Vector Register (IVECT) or write to the Interrupt Request Mask Register (IMASK) in the EIT handler unless interrupts are disabled.
Note 3: When multiple interrupts are disabled, execute processing in [4]. Processing in [4] is unnecessary if multiple interrupts are enabled by executing processing in [6] and [9].
Note 4: To enable multiple interrupts, execute processing in [6] and [9].
Note 5: To reenable interrupts (by setting the IE bit to "1") after reading the Interrupt Vector Register (IVECT), perform a dummy access to the internal memory, etc. before reenabling interrupts. In the example here, there is no need to add a dummy access because the ICU vector table is read after reading the IVECT register. Similarly, to reenable interrupts (by setting the IE bit to "1") after writing to the Interrupt Request Mask Register (IMASK), perform a dummy access to the internal memory, etc. before reenabling interrupts.
Note 6: Depend on groups of microcomputer. the address assigned by ICU Vector is different. (Here, 32170 is assumed to be an example.)

Figure 4.3.1 The example of EI handler processing operation from internal peripheral I/O of M32R/ECU

The External Interrupt (EI) in Figure 4.3.1 is explained here.

(1) Saving each registers to the stack
(BPC, PSW, general-purpose registers, and accumulator)

- Backup PC (BPC) evacuates the value of a program counter (PC) when EIT occurs.
  At that time, the value of PC just before that or next commanded is set, and the value of BPC is returned to PC when "RTE command" runs.
  (Lines 147,162 in Chapter 5, "Sample Programming Code")
- PSW is the register that displays the status of M32R, and consists of PSW field usually used and BPSW field for evacuating PSW field when EIT occurs. SM, IE, and C in PSW register are evacuated respectively to BPSW, when EIT occurs.
  (Lines 145,161 in Chapter 5, "Sample Programming Code")
- General-purpose registers (R0 - R15) that are used in the EIT handler must always be saved. The registers used in user processing executed from the EIT handler also need to be saved.
  (Lines 129 through 132,134,144,146,148,156,157 in Chapter 5, "Sample Programming Code")
- The accumulator (ACC) must always be saved if DSP feature instructions or the multiply instruction (MUL) are used in the EIT handler.
  (Lines 137 through 140 in Chapter 5, "Sample Programming Code")

(2) Reading out the Interrupt Mask Register (IMASK) and saving to the stack
(Lines 133,142,163 in Chapter 5, "Sample Programming Code")

- Reading out the Interrupt Mask Register and saving to the stack

(3) Reading out the Interrupt Vector Register (IVECT)
(Lines 143,150 in Chapter 5, "Sample Programming Code")

- Reading out the IVECT.
- By reading out this IVECT, hardware set the interrupt priority level (ILEVEL) of the interrupt demand sources received automatically to IMASK register as a new value.

(4) Reading out and overwrite IMASK
(Lines 152 to 154 in Chapter 5, "Sample Programming Code")

- Reading out IMASK and overwrite IMASK with the read value.
- This processing is not necessary in case of permitting multiple interruptions.

(5) Reading out the ICU vector table
(Lines 159 in Chapter 5, "Sample Programming Code")

- Read out the ICU vector table received interrupt demand sources. The address corresponding the ICU vector table is obtained by zero expansion that expands the contents read in (3) above (address low 16 bits of the ICU vector table which receives interrupt demand sources) of the interrupt vector table register.

(6) Permission of multiplex interrupt
(Lines 135,165 in Chapter 5, "Sample Programming Code")

- In case of permitting higher interrupt of a priority level during interrupt processing (permission of multiplex interrupt), it sets IE bit of PSW to "1". In this sample programming code, multiplex interrupt is set as permission.

(7) Branching to Interrupt handler of each Internal Peripheral I/Os
(Lines 167 in Chapter 5, "Sample Programming Code")

- It branches to the start address of the interrupt handler read in (5) above.

(8) Processing Interrupt handler of each Internal Peripheral I/Os

(9) Prohibiting interrupt
(Lines 169,170 in Chapter 5, "Sample Programming Code")

- By clearing IE bit to "0", Interrupt is prohibited.

(10) Restoring the Interrupt Mask Register
(Lines 172,173,178 in Chapter 5, "Sample Programming Code")

- Restore the Interrupt Mask Register that is saved in (2).

(11) Restoring each registers from the stack
(Accumulator, general-purpose registers, PSW, BPC.)
(Lines 171,172,174 through 177, 179 through 195 in Chapter 5, "Sample Programming Code")

- Restore the Register that is saved in (1).

(12) Completion of external interrupt processing
(Lines 197 in Chapter 5, "Sample Programming Code")

- Complete external interrupt processing by RTE command.

4.4 Interrupts from Internal Peripheral I/Os

Interrupts from the internal peripheral I/Os are taken care of by the Interrupt Controller (ICU) along with system break interrupts, and are notified as external interrupt (EI) to the M32R CPU.
The 32170 has a total of 31 sources for interrupts from the internal peripheral I/Os, which are assigned priority in levels up to 8 (including interrupts disabled) for management purposes. When multiple interrupts with the same priority occur simultaneously, they are resolved by predetermined hardware priority. The source for an interrupt request generated from internal peripheral I/Os can be identified by reading the internal peripheral I/Os' interrupt status register.

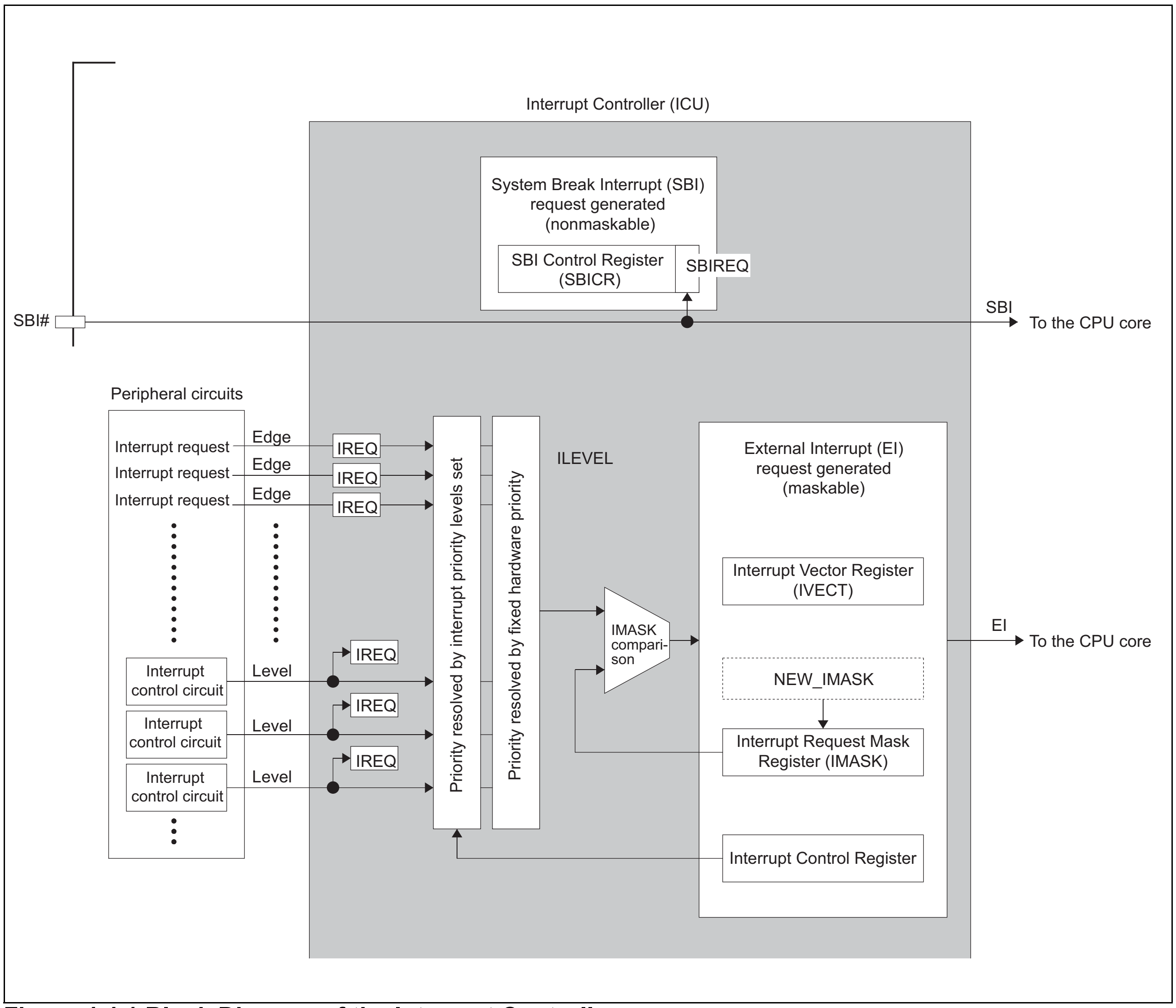

Interrupt Controller (ICU)
System Break Interrupt (SBI) request generated (nonmaskable)
SBI Control Register (SBICR)
SBIREQ
SBI#
SBI
To the CPU core
Peripheral circuits
Interrupt request
Interrupt request
Interrupt request
Edge
Edge
Edge
IREQ
IREQ
IREQ
Interrupt control circuit
Interrupt control circuit
Interrupt control circuit
Level
Level
Level
IREQ
IREQ
IREQ
Priority resolved by interrupt priority levels set
Priority resolved by fixed hardware priority
ILEVEL
IMASK comparison
External Interrupt (EI) request generated (maskable)
Interrupt Vector Register (IVECT)
NEW_IMASK
Interrupt Request Mask Register (IMASK)
Interrupt Control Register
EI
To the CPU core

Figure 4.4.1 Block Diagram of the Interrupt Controller

4.4.1 Interrupt Sources of Internal Peripheral I/Os

The interrupt controller receives as its inputs the interrupt requests from MJT (multijunction timer), DMAC, serial interface, A/D converter, RTD, and CAN.
For details about these interrupts, refer each section at user's manual in which the relevant internal peripheral I/O is described.

Table 4.4.1 32170 Interrupt Sources of Internal Peripheral I/Os

Interrupt source	Contents	Number of Input Source	ICU Type of Input Source (Note 1)
A/D0 converter interrupt	Single-shot operation of A/D0 converter's scan mode completed, single mode completed, or comparator mode completed	1	Edge-type
A/D1 converter interrupt	Single-shot operation of A/D1 converter's scan mode completed, single mode completed, or comparator mode completed	1	Edge-type
SIO0 transmit interrupt	SIO0 transmit buffer empty interrupt	1	Edge-type
SIO0 receive interrupt	SIO0 reception completed or receive error interrupt	1	Edge-type
SIO1 transmit interrupt	SIO1 transmit buffer empty interrupt	1	Edge-type
SIO1 receive interrupt	SIO1 reception completed or receive error interrupt	1	Edge-type
SIO2, 3 transmit/ receive interrupt	SIO2, 3 reception completed or receive error interrupt, Transmit buffer empty	4	Level-type
SIO4, 5 transmit/ receive interrupt	SIO4, 5 reception completed or receive error interrupt, Transmit buffer empty	4	Level-type
TID0 output interrupt	TID0 output	1	Edge-type
TID1 output interrupt	TID1 output	1	Edge-type
TID2 output interrupt	TID2 output	1	Edge-type
TOD0 output interrupt	TOD0_0 - TOD0_7 outputs	8	Level-type
TOD1+TOM0 output Interrupt	TOD1_0 - TOD1_7 outputs + TOM0_0 - TOM0_7 outputs	16	Level-type
TML1 input interrupt	TML1 input (TIN30 - TIN33 inputs)	4	Level-type
RTD interrupt	RTD interrupt generation command	1	Edge-type
DMA transfer interrupt 0	DMA0 - 4 transfers completed	5	Level-type
DMA transfer interrupt 1	DMA5 - 9 transfers completed	5	Level-type
CAN0 transmit/receive & error interrupt	CAN0 transmission completed, CAN0 reception completed CAN0 error passive, CAN0 error bus off, CAN0 bus error	19	Level-type
MJT output interrupt 7	MJT output interrupt group 7 (TMS0 and TMS1 outputs)	2	Level-type
MJT output interrupt 6	MJT output interrupt group 6 (TOP8 and TOP9 outputs)	2	Level-type
MJT output interrupt 5	MJT output interrupt group 5 (TOP10 outputs)	1	Edge-type
MJT output interrupt 4	MJT output interrupt group 4 (TIO4 - TIO7 outputs)	4	Level-type
MJT output interrupt 3	MJT output interrupt group 3 (TIO8 and TIO9 outputs)	2	Level-type
MJT output interrupt 2	MJT output interrupt group 2 (TOP0 - TOP5 outputs)	6	Level-type
MJT output interrupt 1	MJT output interrupt group 1 (TOP6 and TOP7 outputs)	2	Level-type
MJT output interrupt 0	MJT output interrupt group 0 (TIO0 - TIO3 outputs)	4	Level-type
MJT input interrupt 4	MJT input interrupt group 4 (TIN3 - TIN6 inputs)	4	Level-type
MJT input interrupt 3	MJT input interrupt group 3 (TIN20 - TIN23 inputs)	4	Level-type
MJT input interrupt 2	MJT input interrupt group 2 (TIN12 - TIN19 inputs)	8	Level-type
MJT input interrupt 1	MJT input interrupt group 1 (TIN0 - TIN2 inputs)	3	Level-type
MJT input interrupt 0	MJT input interrupt group 0 (TIN7 - TIN11 inputs)	5	Level-type

Note 1: ICU type of input source
Edge-type: Interrupt requests are generated on a falling edge of the interrupt signal applied to the ICU.
Level-type: Interrupt requests are generated when the interrupt signal applied to the ICU is held low.
For these level-recognized interrupts, the ICU's Interrupt Control Register IRQ bit cannot be set or cleared in software.

Interrupts signaled to the ICU are recognized by either edge or level. Each type of interrupt is schematically explained below.

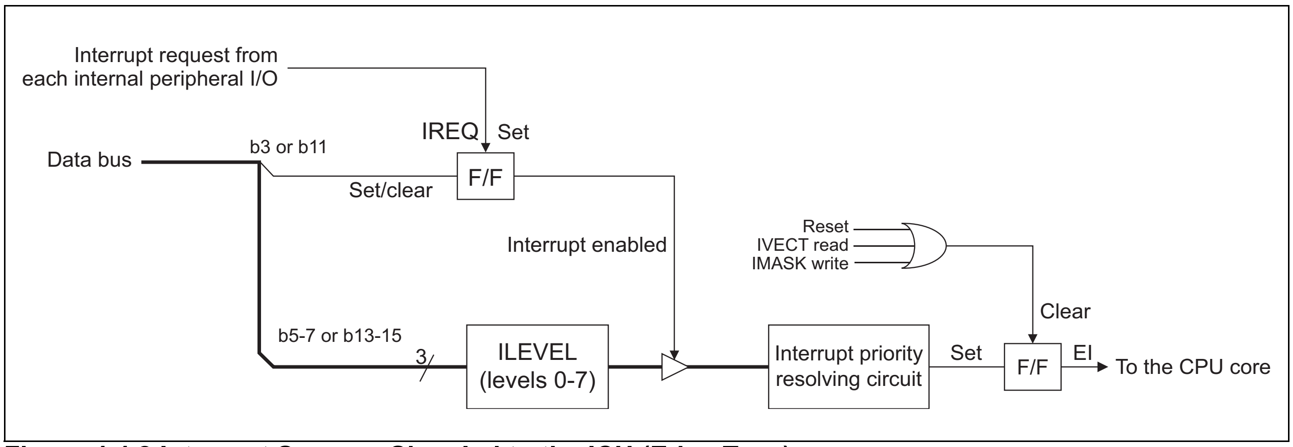


Figure 4.4.2 Interrupt Sources Signaled to the ICU (Edge Type)

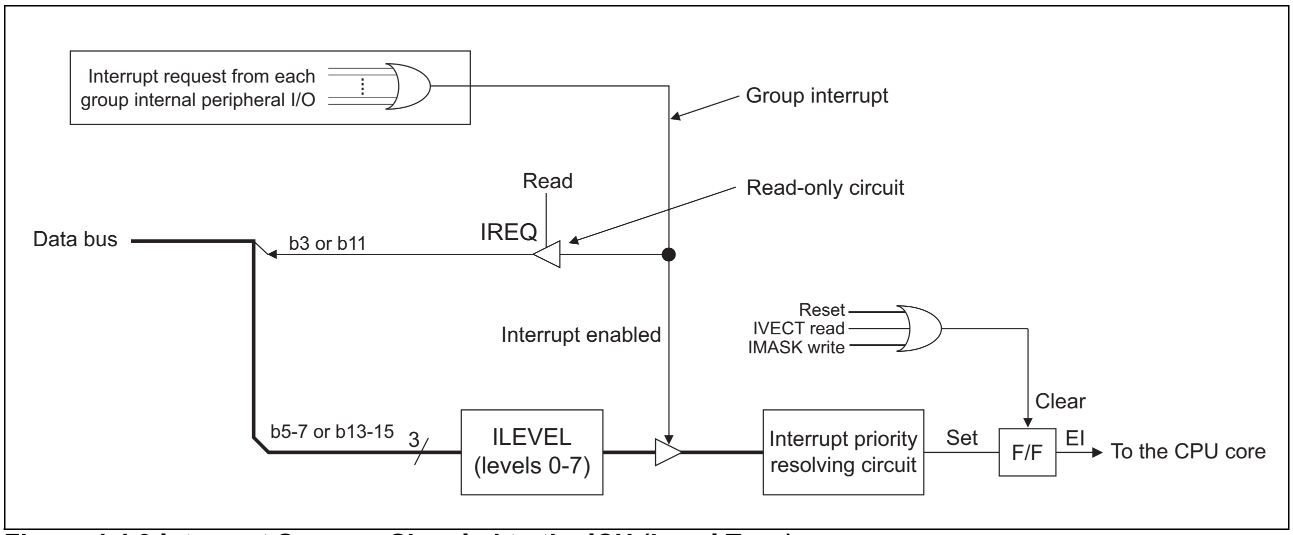


Figure 4.4.3 Interrupt Sources Signaled to the ICU (Level Type)

(1) Edge-type interrupt sources

For edge-type interrupt sources, the ICU's internal Interrupt Control Register interrupt request bit can be set and cleared by writing to the register in software. The bit can also be cleared by reading out the Interrupt Vector Register (IVECT).

(2) Level-type interrupt sources

The level-type interrupt sources differ from the other type in that two or more interrupt sources are grouped into a single source as they are mapped to the interrupt vector table. These interrupt sources are controlled for interrupt request and resolved for interrupt input by an interrupt control register in each internal peripheral I/O before being fed into the ICU. Therefore, the ICU's internal interrupt request bits function only as an interrupt-enabled interrupt request judgment bit and cannot be set or reset by writing in software. The diagram below shows example configuration of Interrupt request status register and Mask register.

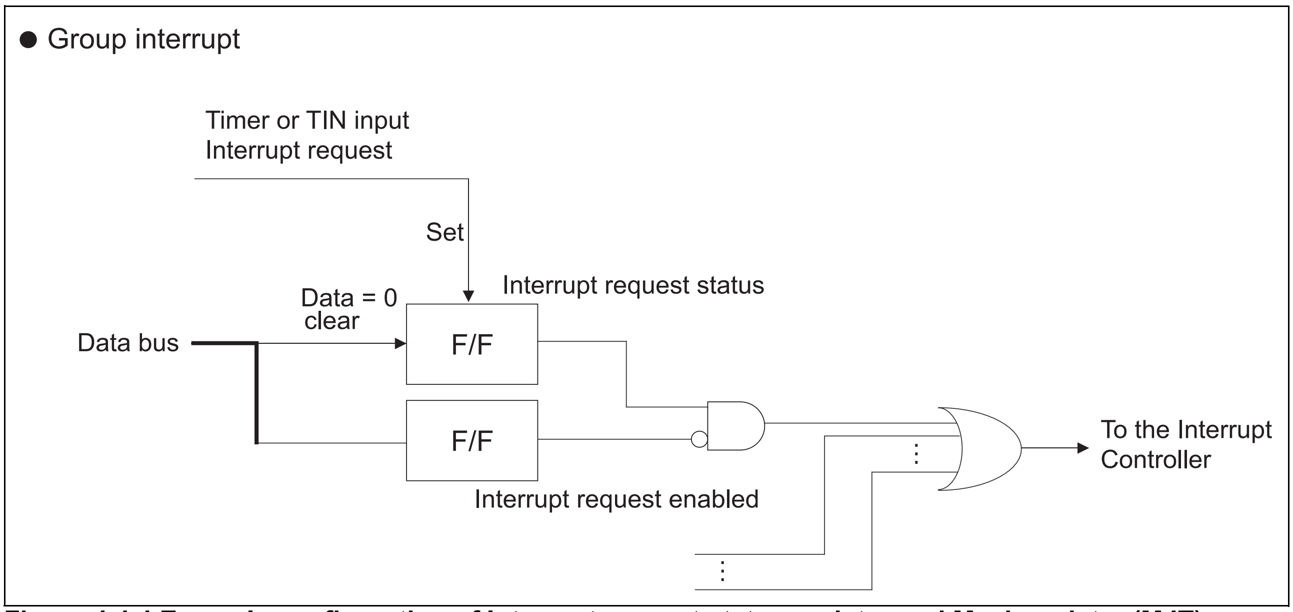
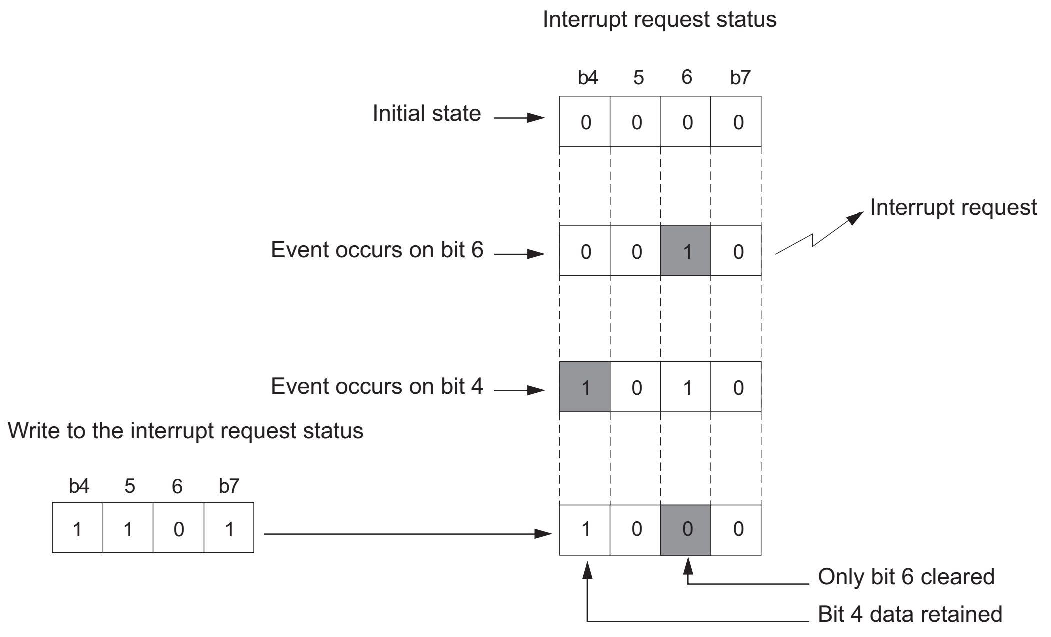


Figure 4.4.4 Example configuration of Interrupt request status register and Mask register (MJT)

- Internal peripheral I/O interrupt request status bit

This status bit is used to determine whether any interrupt is requested. This bit is set in hardware when an interrupt request occurs, but cannot be set in software. The status bit is cleared by writing as “0” but does not change its state by writing as “1”. Because this bit behaves independently of interrupt mask bits, it can also be used to verify the operation of peripheral functions. During interrupt processing, make sure that of the grouped interrupt request status flags, only the flag for which interrupt has been serviced is cleared. Clearing any flag for which interrupt has not been serviced yet results in the pending interrupt requests being also cleared. When all interrupt request status flags for grouped interrupt sources are cleared, the level-type interrupt request for that interrupt group is cleared.

● Example for clearing interrupt request status



● Program example

• To clear the Interrupt Request Status Register 0 (ISTREG) interrupt request status 1, ISTAT1 (0x02 bit)

○
```
ISTREG = 0xfd;          /* Clear ISTAT1 (0x02 bit) only */
```

To clear an interrupt request status, always be sure to write "1" to all other interrupt request status bits. At this time, avoid using a logic operation like the one shown below. Because it requires three step-ISTREG read, logic operation and write, if another interrupt request occurs between the read and write, status may be inadvertently cleared.

✕
```
ISTREG &= 0xfd;         /* Clear ISTAT1 (0x02 bit) only */
```

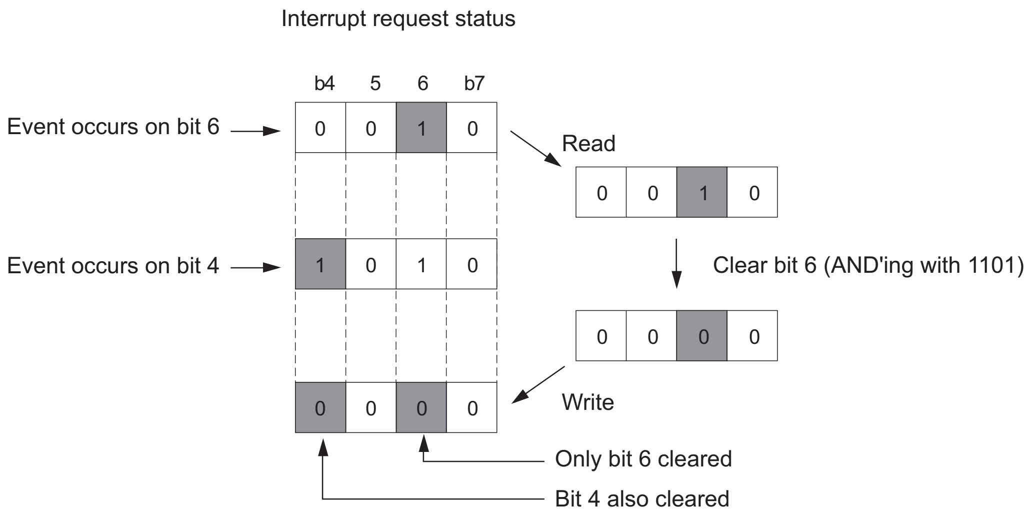


Figure 4.4.5 Example for Clearing Interrupt Status

5. Sample Programming Code

The sample program of a startup program (startup.ms) is shown below.
Depending on the contents of an application program, there may be some processing that should be added or deleted besides the following processing,

```
;""FILE COMMENT"" ****************************************************
;*      M32R C Programming                      Rev. 1.01
;*               < Sample startup Program for 32170/32171/32174/32176 >
;*
;*      Copyright (c) 2003 Renesas Technology Corporation
;*                         And Renesas Solutions Corporation
;*                         All Rights Reserved
;**********************************************************************
;
;****************************************
; EIT Vector Entry
;****************************************
;
        .SECTION        EITVECT, CODE, ALIGN=4
;
        .EXPORT reset, EIT_reset, EIT_loop
;
reset:
        BRA             EIT_reset:24            ; H'0000 0000 Reset Interrupt (RI)
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BRA             EIT_loop:24             ; H'0000 0010 System Break Interrupt (SBI)
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BRA             EIT_loop:24             ; H'0000 0020 Reserved Instruction Exception (RIE)
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BRA             EIT_loop:24             ; H'0000 0030 Address Exception (AE)
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BRA             EIT_loop:24             ; H'0000 0040   Trap 0
        BRA             EIT_loop:24             ; H'0000 0044   Trap 1
        BRA             EIT_loop:24             ; H'0000 0048   Trap 2
        BRA             EIT_loop:24             ; H'0000 004C   Trap 3
        BRA             EIT_loop:24             ; H'0000 0050   Trap 4
        BRA             EIT_loop:24             ; H'0000 0054   Trap 5
        BRA             EIT_loop:24             ; H'0000 0058   Trap 6
        BRA             EIT_loop:24             ; H'0000 005C   Trap 7
        BRA             EIT_loop:24             ; H'0000 0060   Trap 8
        BRA             EIT_loop:24             ; H'0000 0064   Trap 9
        BRA             EIT_loop:24             ; H'0000 0068   Trap 10
        BRA             EIT_loop:24             ; H'0000 006C   Trap 11
        BRA             EIT_loop:24             ; H'0000 0070   Trap 12
        BRA             EIT_loop:24             ; H'0000 0074   Trap 13
        BRA             EIT_loop:24             ; H'0000 0078   Trap 14
        BRA             EIT_loop:24             ; H'0000 007C   Trap 15
        BRA             EIT_ei:24               ; H'0000 0080   External Interrupt (EI)
;
        .SECTION        PROTECTID, DATA, ALIGN=1
        .DATA.B         H'FF,H'FF,H'FF,H'FF,H'FF,H'FF,H'FF,H'FF ; H'0000 0084 Protect ID
        .DATA.B         H'FF,H'FF,H'FF,H'FF,H'FF,H'FF,H'FF,H'FF ;
;
;
;****************************************
; ICU Vector Table
;****************************************
;
        .SECTION        ICUVECT, DATA, ALIGN=4
;
;
vectbl:
        .DATA.W         EIT_reset               ; H'0000 0094   MJT Input Interrupt 4:TIN3-TIN6
        .DATA.W         EIT_reset               ; H'0000 0098   MJT Input Interrupt 3:TIN20-TIN23
        .DATA.W         EIT_reset               ; H'0000 009C   MJT Input Interrupt 2:TIN12-TIN19
        .DATA.W         EIT_reset               ; H'0000 00A0   MJT Input Interrupt 1:TIN0-TIN2
        .DATA.W         EIT_reset               ; H'0000 00A4   MJT Input Interrupt 0:TIN7-TIN11
        .DATA.W         EIT_reset               ; H'0000 00A8   MJT Output Interrupt 7:TMS0,TMS1
        .DATA.W         EIT_reset               ; H'0000 00AC   MJT Output Interrupt 6:TOP8,TOP9
        .DATA.W         EIT_reset               ; H'0000 00B0   MJT Output Interrupt 5:TOP10
        .DATA.W         EIT_reset               ; H'0000 00B4   MJT Output Interrupt 4:TIO4-TIO7
        .DATA.W         EIT_reset               ; H'0000 00B8   MJT Output Interrupt 3:TIO8,TIO9
        .DATA.W         EIT_reset               ; H'0000 00BC   MJT Output Interrupt 2:TOP0-TOP5
        .DATA.W         EIT_reset               ; H'0000 00C0   MJT Output Interrupt 1:TOP6,TOP7
        .DATA.W         EIT_reset               ; H'0000 00C4   MJT Output Interrupt 0:TIO0-TIO3
        .DATA.W         EIT_reset               ; H'0000 00C8   DMAC0-4 Interrupt:DMA0-DMA4
        .DATA.W         EIT_reset               ; H'0000 00CC   SIO1 Receive Interrupt
        .DATA.W         EIT_reset               ; H'0000 00D0   SIO1 Transmit Interrupt
        .DATA.W         EIT_reset               ; H'0000 00D4   SIO0 Receive Interrupt
        .DATA.W         EIT_reset               ; H'0000 00D8   SIO0 Transmit Interrupt
        .DATA.W         EIT_reset               ; H'0000 00DC   A-D0 Conversion Interrupt
        .DATA.W         EIT_reset               ; H'0000 00E0   TID0 Output Interrupt
        .DATA.W         EIT_reset               ; H'0000 00E4   TOD0 Output Interrupt
        .DATA.W         EIT_reset               ; H'0000 00E8   DMAC5-9 Interrupt:DMA5-DMA9
        .DATA.W         EIT_reset               ; H'0000 00EC   SIO2,3 Transmit/Receive Interrupt
        .DATA.W         EIT_reset               ; H'0000 00F0   RTD Interrupt
        .DATA.W         EIT_reset               ; H'0000 00F4   TID1 Output Interrupt
        .DATA.W         EIT_reset               ; H'0000 00F8   TOD1,TOM0 Output Interrupt
        .DATA.W         EIT_reset               ; H'0000 00FC   SIO4,5 Transmit/Receive Interrupt
        .DATA.W         EIT_reset               ; H'0000 0100   A-D1 Conversion Interrupt
        .DATA.W         EIT_reset               ; H'0000 0104   TID2 Output Interrupt
        .DATA.W         EIT_reset               ; H'0000 0108   TML1 Input Interrupt
        .DATA.W         EIT_reset               ; H'0000 010C   CAN0 Transmit/Receive & Error Interrupt
        .DATA.W         EIT_reset               ; H'0000 0110   CAN1 Transmit/Receive & Error Interrupt
;
;*****************************************
; Set SFR Address
;*****************************************
;
        .SECTION        SFR, DATA, ALIGN=1
        .RES.B          H'4000
;
IMASK           .EQU    H'00800004      ; IMASK Address
;
;*****************************************
; External Interrupt Handler
;*****************************************
;
        .SECTION        EIT_P, CODE, ALIGN=4
;
        .EXPORT EIT_ei
;
EIT_ei:
        ST              R0,@-R15                ; PUSH  R0                              (R0)
;
        ST              R1,@-R15                ; PUSH  R1                              (R1)
        ST              R2,@-R15                ; PUSH  R2                              (R2)
        LD24            R0,#IMASK               ;
        ST              R3,@-R15                ; PUSH  R3                              (R3)
        LDI             R3,#H'40                ;
;
        MVFACHI         R1                      ; PUSH  Accumulator
        MVFACLO         R2                      ;
        ST              R1,@-R15                ;                                       (AccH)
        ST              R2,@-R15                ;                                       (AccL)
;
        LDB             R2,@R0                  ; Read IMASK(H'0080 0004) Register
        ADDI            R0,#-4                  ;
        ST              R4,@-R15                ; PUSH  R4                              (R4)
        MVFC            R4,PSW                  ;
        ST              R5,@-R15                ; PUSH  R5                              (R5)
        MVFC            R5,BPC                  ;
        ST              R6,@-R15                ; PUSH  R6                              (R6)
;
        LDH             R1,@R0                  ; Read IVECT(H'0080 0000) Register
;
        LD24            R0,#IMASK               ; Overwrite IMASK Register
        LDB             R6,@R0                  ;
        STB             R6,@R0                  ;
;
        ST              R7,@-R15                ; PUSH  R7                              (R7)
        ST              R14,@-R15               ; PUSH  Link Register                   (R14)
;
        LD              R1,@R1                  ; Read ICU Vector Table
;
        ST              R4,@-R15                ; PUSH  PSW                             (PSW)
        ST              R5,@-R15                ; PUSH  BPC                             (BPC)
        ST              R2,@-R15                ; PUSH  IMASK Register                  (IMASK)
;
        MVTC            R3,PSW                  ; Enable Interrupt
;
```

```
        JL              R1                      ; Call Interrupt Handler
;
        LDI             R5,#H'00                ;
        MVTC            R5,PSW                  ; Disable Interrupt
;
        LD24            R3,#IMASK               ;
        LD              R2,@R15+                ; POP   IMASK                   (IMASK)
        LD              R1,@R15+                ; POP   BPC                     (BPC)
        LD              R0,@R15+                ; POP   PSW                     (PSW)
;
        LD              R14,@R15+               ; POP   Link Register           (R14)
        STB             R2,@R3                  ;
        LD              R7,@R15+                ; POP   R7                      (R7)
        MVTC            R1,BPC                  ;
        LD              R6,@R15+                ; POP   R6                      (R6)
        LD              R5,@R15+                ; POP   R5                      (R5)
        LD              R4,@R15+                ; POP   R4                      (R4)
;
        LD              R2,@R15+                ; POP Accumulator               (AccL)
        LD              R1,@R15+                ;                               (AccH)
        MVTC            R0,PSW                  ;
        MVTACLO         R2                      ;
        MVTACHI         R1                      ;
;
        LD              R3,@R15+                ; POP   R3                      (R3)
        LD              R2,@R15+                ; POP   R2                      (R2)
        LD              R1,@R15+                ; POP   R1                      (R1)
;
        LD              R0,@R15+                ; POP   R0                      (R0)
;
        RTE
;
;****************************************
; EIT_loop
;****************************************
;
EIT_loop:
        BRA             EIT_loop
;
;****************************************
; Start Up Program
;****************************************
;
        .IMPORT $main
;
        .SECTION        C, DATA, ALIGN=4
        .SECTION        D, DATA, ALIGN=4
        .SECTION        B, DATA, ALIGN=4
        .SECTION        ROM_D, DATA, ALIGN=4
;
;++++++++++++++++++++++++++++++++++++++++
; Set Interrupt Stack
;
        .SECTION        SPINT, DATA, ALIGN=4
        .RES.B          2048                    ; Interrupt Stack Area
;
;++++++++++++++++++++++++++++++++++++++++
; Set User Stack
;
        .SECTION        SPUSR, DATA, ALIGN=4
;       .RES.B          2048                    ; User Stack Area
;
;++++++++++++++++++++++++++++++++++++++++
; Startup & Exit
;
        .SECTION        P, CODE, ALIGN=4
EIT_reset:
        LDI             R0, #H'00               ;
        MVTC            R0, PSW                 ; Disable Interrupt
        LDI             R0, #7                  ;
        LD24            R1, #IMASK              ;
        STB             R0, @R1                 ;
;
;       LD24            R1, #(SPUSR+sizeof(SPUSR))      ;
        LD24            R2, #(SPINT+sizeof(SPINT))      ;
;       MVTC            R1, SPU                 ; Set User Stack Pointer
        MVTC            R2, SPI                 ; Set Interrupt Stack Pointer
;
;++++++++++++++++++++++++++++++++++++++++
; Clear B Section
;
        LD24            R5, #sizeof(B)
        BLEZ            R5, loop_cnt0
        LD24            R4, #B
        LDI             R0, #0
loop0:
        STB             R0, @R4
        ADDI            R4, #1
        ADDI            R5, #-1
        BNEZ            R5, loop0
loop_cnt0:
;
;++++++++++++++++++++++++++++++++++++++++
; Data Set (ROM_D Section => D Section)
;
        LD24            R6, #sizeof(ROM_D)
        BLEZ            R6, loop_cnt1
        LD24            R4, #D
        LD24            R5, #ROM_D
loop1:
        LDB             R0, @R5
        STB             R0, @R4
        ADDI            R4, #1
        ADDI            R5, #1
        ADDI            R6, #-1
        BNEZ            R6, loop1
loop_cnt1:
;
;++++++++++++++++++++++++++++++++++++++++
; Set Base Register
;
;       .EXPORT         __REL_BASE13
        .EXPORT         __REL_BASE12
;       .EXPORT         __REL_BASE11
;__REL_BASE13           .EQU    0x00808000
__REL_BASE12            .EQU    0x00808000
;__REL_BASE11           .EQU    0x00808000
;
;       SETH            R13, #HIGH(__REL_BASE13)
;       OR3             R13, R13, #LOW(__REL_BASE13)
;
        SETH            R12, #HIGH(__REL_BASE12)
        OR3             R12, R12, #LOW(__REL_BASE12)
;
;       SETH            R11, #HIGH(__REL_BASE11)
;       OR3             R11, R11, #LOW(__REL_BASE11)
;
;++++++++++++++++++++++++++++++++++++++++
; Call main()
;
;       LDI             R0, #H'C0               ; Enable Interrupt, Use User Stack
        LDI             R0, #H'40               ; Enable Interrupt, Use Interrupt Stack
        MVTC            R0, PSW
;
        BL              $main                   ; Call C(main) Routine
;
;++++++++++++++++++++++++++++++++++++++++
;
endless:
        BRA             endless                 ; Dummy
;
;++++++++++++++++++++++++++++++++++++++++
; Set/Clear IE Flag Routine
;
        .EXPORT                 $EnInt
        .EXPORT                 $DisInt
$EnInt:
        MVFC            R0, PSW
        OR3             R0, R0, #H'0040
        MVTC            R0, PSW
        JMP             R14
;
$DisInt:
        MVFC            R0, PSW
        AND3            R0, R0, #H'FFBF
        MVTC            R0, PSW
        JMP             R14
;
        .END
```

6. Reference of Document

- 32170/32174 Group User's Manual Rev.2.10
- 32171 Group User's Manual Rev.2.00
- 32176 Group User's Manual Rev.1.01
- M32R Family Software Manual Rev.1.20
- M3T-CC32R V.4.30 User's Manual (Compiler)
- M3T-AS32R V.4.30 User's Manual (Assembler)

(Please get the latest one from Renesas Technology Corp. website.)

7. Website and Support Center

- Renesas Technology Corp. website
  http://www.renesas.com/

- Customer Support Center for all Products and Technical Support Center for M32R Family
  Customer Support Center: csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jan 13, 2006	–	First Edition issued