

Section 5 Interrupt Controller

5.1 Overview

The interrupt controller decides which interrupts to accept, and how to deal with multiple interrupts. It also decides whether an interrupt should be served by the CPU or by the data transfer controller (DTC). This section explains the features of the interrupt controller, describes its internal structure and control registers, and details the handling of interrupts.

For detailed information on the data transfer controller, see section 6, “Data Transfer Controller.”

5.1.1 Features

Three main features of the interrupt controller are:

- Interrupt priorities are user-programmable.
User programs can set priority levels from 7 (high) to 0 (low) in four interrupt priority (IPR) registers for IRQ0, IRQ1, and each of the on-chip supporting modules—for every interrupt, that is, except the nonmaskable interrupt (NMI). NMI has the highest priority level (8) and is normally always accepted. An interrupt with priority level 0 is always masked.
- Multiple interrupts on the same level are served in a default priority order.
Lower-priority interrupts remain pending until higher-priority interrupts have been handled.
- For most interrupts, software can select whether to have the interrupt served by the CPU or the on-chip data transfer controller (DTC).
User programs can make this selection by setting and clearing bits in four data transfer enable (DTE) registers. The data transfer controller can be started by any interrupts except NMI, the error interrupt (ERI) from the on-chip serial communication interface, and the overflow interrupts (FOVI and OVI) from the on-chip timers.

5.1.2 Block Diagram

Figure 5-1 shows the block configuration of the interrupt controller.

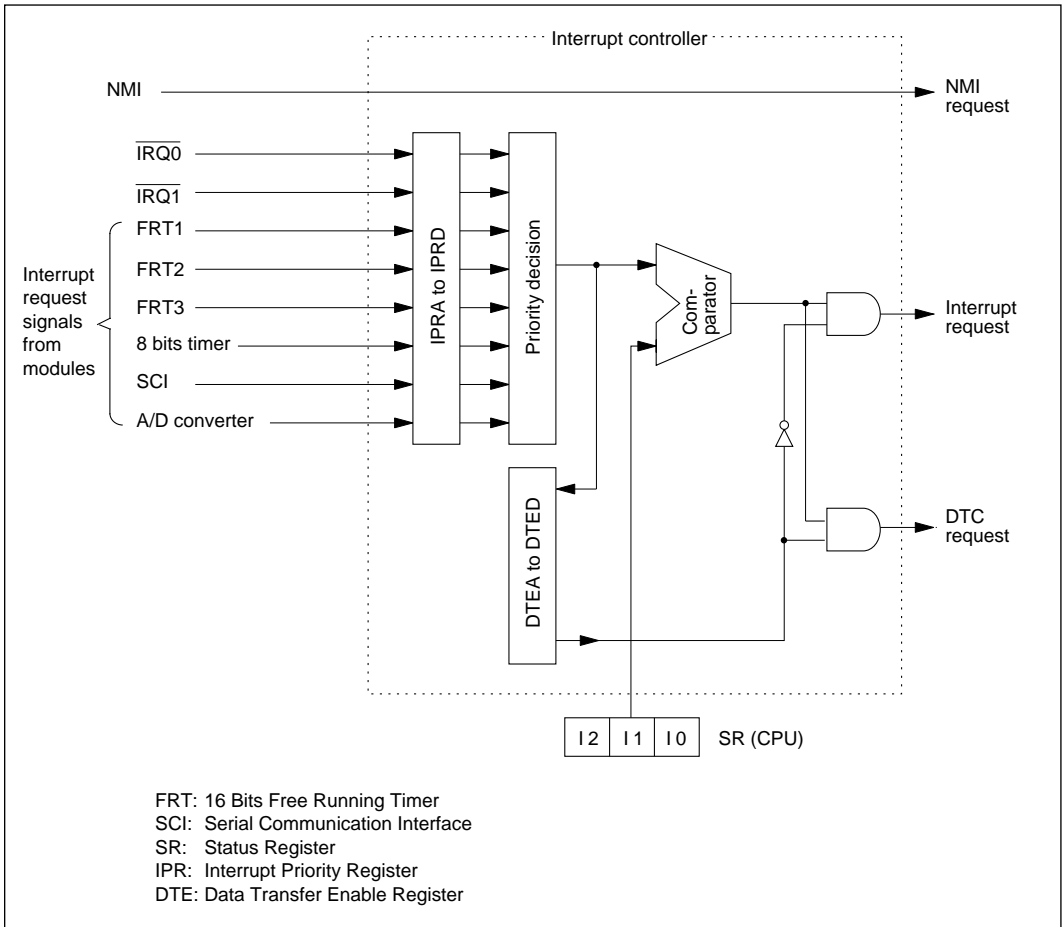


Figure 5-1 Interrupt Controller Block Diagram

5.1.3 Register Configuration

The four interrupt priority registers (IPRA to IPRD) and four data transfer enable registers (DTEA to DTED) are 8-bit registers located at addresses H'FFF0 to H'FFF7 in the register field in page 0 of the address space. Table 5-1 lists their attributes.

Table 5-1 Interrupt Controller Registers

| Name | | Abbreviation | Read/Write | Address | Initial Value |
|-------------------------------|---|--------------|------------|---------|---------------|
| Interrupt priority register | A | IPRA | R/W | H'FFF0 | H'00 |
| | B | IPRB | R/W | H'FFF1 | H'00 |
| | C | IPRC | R/W | H'FFF2 | H'00 |
| | D | IPRD | R/W | H'FFF3 | H'00 |
| Data transfer enable register | A | DTEA | R/W | H'FFF4 | H'00 |
| | B | DTEB | R/W | H'FFF5 | H'00 |
| | C | DTEC | R/W | H'FFF6 | H'00 |
| | D | DTED | R/W | H'FFF7 | H'00 |

5.2 Interrupt Types

There are 22 distinct types of interrupts: 3 external interrupts originating off-chip and 19 internal interrupts originating in the on-chip supporting modules.

5.2.1 External Interrupts

The three external interrupts are NMI, IRQ0, and IRQ1.

NMI (NonMaskable Interrupt): This interrupt has the highest priority level (8) and cannot be masked. An NMI is generated by input to the NMI pin, and can also be generated by a watchdog timer (WDT) overflow. The input at the NMI pin is edge-sensed. A user program can select whether to have the interrupt occur on the rising edge or falling edge of the NMI input by setting or clearing the nonmaskable interrupt edge bit (NMIEG) in the port 1 control register (P1CR).

In the NMI exception-handling sequence, the T (Trace) bit in the CPU status register (SR) is cleared to “0,” and the interrupt mask level in I2 to I0 is set to 7, masking all other interrupts. The interrupt controller holds the NMI request until the NMI exception-handling sequence begins, then clears the NMI request, so if another interrupt is requested at the NMI pin during the NMI exception-handling sequence, the NMI exception-handling sequence will be carried out again.

A watchdog timer overflow generates an NMI if the TME and WT/ $\overline{\text{T}}$ bits in the watchdog timer's status/control register are both set to “1.” See section 13, “Watchdog Timer” for details.

Coding Examples:

To select the rising edge of the NMI input: `BSET.B #4, @H'FFFC`

To select the falling edge of the NMI input: `BCLR.B #4, @H'FFFC`

IRQ0 (Interrupt Request 0): An IRQ0 interrupt can be requested by a Low input to the $\overline{\text{IRQ0}}$ pin and/or a watchdog timer overflow. A Low $\overline{\text{IRQ0}}$ input requests an IRQ0 interrupt if the interrupt request enable 0 bit (IRQ0E) in the PICR is set to “1.” $\overline{\text{IRQ0}}$ must be held Low until the CPU accepts the interrupt. Otherwise the request will be ignored. A watchdog timer overflow requests an IRQ0 interrupt if the TME bit is set to “1” and the WT/IT bit is cleared to “0” in the watchdog timer's control/status register. See section 13, “Watchdog Timer” for details of the watchdog timer.

The IRQ0 interrupt can be assigned any priority level from 7 to 0 by setting the corresponding value in the upper four bits of IPRA. If bit 4 of data transfer enable register A (DTEA) is set to “1,” an IRQ0 interrupt starts the data transfer controller. Otherwise the interrupt is served by the CPU.

In the CPU interrupt-handling sequence for IRQ0, the T bit of the status register is cleared to “0,” and the interrupt mask level is set to the value in the upper four bits of IPRA.

Coding Examples:

To enable IRQ0 to be requested by $\overline{\text{IRQ0}}$ input: `BSET.B #5, @H'FFFC`

To assign priority level 7 to IRQ0: `OR.B #70, @H'FFF0`

To have IRQ0 start the DTC: `BSET.B #4, @H'FFF4`

$\overline{\text{IRQ1}}$ (Interrupt Request 1): An IRQ0 interrupt is requested by a High-to-Low transition at the $\overline{\text{IRQ1}}$ pin. The IRQ1 interrupt is enabled only when the interrupt request enable 1 bit (IRQ1E) in the PICR is set to “1.”

The IRQ1 interrupt can be assigned any priority level from 7 (high) to 0 (low) by setting the corresponding value in the lower four bits of IPRA. If bit 0 of data transfer enable register A (DTEA) is set to “1,” an IRQ1 interrupt starts the data transfer controller. Otherwise the interrupt is served by the CPU.

The interrupt controller holds the IRQ1 request until the IRQ1 exception-handling sequence begins, then clears the IRQ1 request. If another interrupt is requested at the $\overline{\text{IRQ1}}$ pin during the IRQ1 interrupt-handling routine, the request is held, but the IRQ1 exception-handling sequence is not carried out immediately because the interrupt is masked by bits I2 to I0 in the status register. On return from the interrupt-handling routine one more instruction is executed, then the exception-handling sequence for the second IRQ1 interrupt is carried out.

In the CPU interrupt-handling sequence for IRQ1, the T bit of the CPU status register is cleared to “0,” and the interrupt mask level is set to the value in the lower four bits of IPRA.

Coding Examples:

| | |
|---|---------------------------------|
| To enable IRQ1 to be requested by $\overline{\text{IRQ1}}$ input: | <code>BSET.B #6, @H'FFFC</code> |
| To assign priority level 7 to IRQ0 and level 5 to IRQ1: | <code>MOV.B #75, @H'FFF0</code> |
| To have IRQ1 start the DTC: | <code>BSET.B #0, @H'FFF4</code> |

5.2.2 Internal Interrupts

Nineteen types of internal interrupts can be requested by the on-chip supporting modules. Each interrupt is separately vectored in the exception vector table, so it is not necessary for the user-coded interrupt handler routine to determine which type of interrupt has occurred.

Each of the internal interrupts can be enabled or disabled by setting or clearing an enable bit in the control register of the on-chip supporting module.

An interrupt priority level from 7 to 0 can be assigned to each on-chip supporting module by setting interrupt priority registers B to D. Within each module, different interrupts have a fixed priority order. For most of these interrupts, values set in data transfer enable registers B to D can select whether to have the interrupt served by the CPU or the data transfer controller.


In the CPU interrupt-handling sequence, the T bit of the CPU status register is cleared to “0,” and the interrupt mask level in bits I2 to I0 is set to the value in the IPR.

5.2.3 Interrupt Vector Table

Table 5-2 lists the addresses of the exception vector table entries for each interrupt, and explains how their priority is determined. For the on-chip supporting modules, the priority level set in the interrupt priority register applies to the module as a whole: all interrupts from that module have the same priority level. A separate priority order is established among interrupts from the same module. If the same priority level is assigned to two or more modules and two interrupts are requested simultaneously from these modules, they are served in the priority order indicated in the rightmost column in table 5-2.

A reset clears the interrupt priority registers so that all interrupts except NMI start with priority level 0, meaning that they are unconditionally masked.

Table 5-2 Interrupts, Vectors, and Priorities

| Interrupt | Assignable Priority Levels | | Priority within Module | Vector Table Entry Address | | Priority among Interrupts on Same Level* |
|---------------|----------------------------|---------------------|------------------------|----------------------------|--------------|--|
| | (Initial Level) | IPR Bits | | Minimum Mode | Maximum Mode | |
| NMI | 8 (8) | — | — | H'16 - H'17 | H'2C - H'2F | High |
| IRQ0 | 7 to 0 (0) | IPRA bits 6 to 4 | — | H'40 - H'41 | H'80 - H'83 |  |
| IRQ1 | 7 to 0 (0) | IPRA bits 2 to 0 | — | H'42 - H'43 | H'84 - H'87 | |
| 16-Bit FRT1 | ICI (0) | IPRB bits 6 to 4 | 3 | H'48 - H'49 | H'90 - H'93 | |
| | OCIA | | 2 | H'4A - H'4B | H'94 - H'97 | |
| | OCIB | | 1 | H'4C - H'4D | H'98 - H'9B | |
| | FOVI | | 0 | H'4E - H'4F | H'9C - H'9F | |
| 16-Bit FRT2 | ICI (0) | IPRB bits 2 to 0 | 3 | H'50 - H'51 | H'A0 - H'A3 | |
| | OCIA | | 2 | H'52 - H'53 | H'A4 - H'A7 | |
| | OCIB | | 1 | H'54 - H'55 | H'A8 - H'AB | |
| | FOVI | | 0 | H'56 - H'57 | H'AC - H'AF | |
| 16-Bit FRT3 | ICI (0) | IPRC bits 6 to 4 | 3 | H'58 - H'59 | H'B0 - H'B3 | |
| | OCIA | | 2 | H'5A - H'5B | H'B4 - H'B7 | |
| | OCIB | | 1 | H'5C - H'5D | H'B8 - H'BB | |
| | FOVI | | 0 | H'5E - H'5F | H'BC - H'BF | |
| 8-Bit timer | CMIA (0) | IPRC bits 2 to 0 | 2 | H'60 - H'61 | H'C0 - H'C3 | |
| | CMIB | | 1 | H'62 - H'63 | H'C4 - H'C7 | |
| | OVI | | 0 | H'64 - H'65 | H'C8 - H'CB | |
| SCI | ERI (0) | IPRD bits 6 to 4 | 2 | H'68 - H'69 | H'D0 - H'D3 | |
| | RXI | | 1 | H'6A - H'6B | H'D4 - H'D7 | |
| | TXI | | 0 | H'6C - H'6D | H'D8 - H'DB | |
| A/D converter | ADI (0) | IPRD bits 2 to 0 | — | H'70 - H'71 | H'E0 - H'E3 | Low |

* If two or more interrupts are requested simultaneously, they are handled in order of priority level, as set in registers IPRA to IPRD. If they have the same priority level because they are requested from the same on-chip supporting module, they are handled in a fixed priority order within the module. If they are requested from different modules to which the same priority level is assigned, they are handled in the order indicated in the right-hand column.

5.3 Register Descriptions

5.3.1 Interrupt Priority Registers A to D (IPRA to IPRD)

IRQ₀, IRQ₁, and the on-chip supporting modules are each assigned three bits in one of the four interrupt priority registers (IPRA to IPRD). These bits specify a priority level from 7 (high) to 0 (low) for interrupts from the corresponding source. The drawing below shows the configuration of the interrupt priority registers. Table 5-3 lists their assignments to interrupt sources.

| | | | | | | | | |
|---------------|---|-----|-----|-----|---|-----|-----|-----|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | — | | | | — | | | |
| Initial value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Read/Write | R | R/W | R/W | R/W | R | R/W | R/W | R/W |

Note: Bits 7 and 3 are reserved. They cannot be modified and are always read as “0.”

Table 5-3 Assignment of Interrupt Priority Registers

| Register | Interrupt Request Source | | Address |
|----------|--------------------------|------------------|---------|
| | Bits 6 to 4 | Bits 2 to 0 | |
| IPRA | IRQ ₀ | IRQ ₁ | H'FFF0 |
| IPRB | 16-Bit FRT1 | 16-Bit FRT2 | H'FFF1 |
| IPRC | 16-Bit FRT3 | 8-Bit timer | H'FFF2 |
| IPRD | SCI | A/D converter | H'FFF3 |

As table 5-3 indicates, each interrupt priority register specifies priority levels for two interrupt sources. A user program can assign desired levels to these interrupt sources by writing “000” in bits 6 to 4 or bits 2 to 0 to set priority level 0, for example, or “111” to set priority level 7.

A reset clears registers IPRA to IPRD to H'00, so all interrupts except NMI are initially masked.

When the interrupt controller receives one or more interrupt requests, it selects the request with the highest priority and compares its priority level with the interrupt mask level set in bits I2 to I0 in the CPU status register. If the priority level is higher than the mask level, the interrupt controller passes the interrupt request to the CPU (or starts the data transfer controller). If the priority level is lower than the mask level, the interrupt controller leaves the interrupt request pending until the interrupt mask is altered to a lower level or the interrupt priority is raised. Similarly, if it receives two interrupt requests with the same priority level, the interrupt controller determines their priority as explained in table 5-2 and leaves the interrupt request with the lower priority pending.

5.3.2 Timing of Priority Setting

The interrupt controller requires two system clock (ϕ) periods to determine the priority level of an interrupt. Accordingly, when an instruction modifies an instruction priority register, the new priority does not take effect until after the next instruction has been executed.

5.4 Interrupt Handling Sequence

5.4.1 Interrupt Handling Flow

The interrupt-handling sequence follows the flowchart in figure 5-2. Note that address error, trace exception, and NMI requests bypass the interrupt controller's priority decision logic and are routed directly to the CPU.

1. Interrupt requests are generated by one or more on-chip supporting modules or external interrupt sources.
2. The interrupt controller checks the interrupt priorities set in IPRA to IPRD and selects the interrupt with the highest priority. Interrupts with lower priorities remain pending. Among interrupts with the same priority level, the interrupt controller determines priority as explained in table 5-2.
3. The interrupt controller compares the priority level of the selected interrupt request with the mask level in the CPU status register (bits I2 to I0). If the priority level is equal to or less than the mask level, the interrupt request remains pending. If the priority level is higher than the mask level, the interrupt controller accepts the interrupt request and proceeds to the next step.
4. The interrupt controller checks the corresponding bit (if any) in the data transfer enable registers (DTEA to DTED). If this bit is set to "1," the data transfer controller is started. Otherwise, the CPU interrupt exception-handling sequence is started.

When the data transfer controller is started, the interrupt request is cleared (except for interrupt requests from the serial communication interface, which are cleared by writing to the TDR or reading the RDR).

If the data transfer enable bit is cleared to “0” (or is nonexistent), the sequence proceeds as follows. For the case in which the data transfer controller is started, see section 6, “Data Transfer Controller.”

5. After the CPU has finished executing the current instruction, the program counter and status register (in minimum mode) or program counter, code page register, and status register (in maximum mode) are saved to the stack, leaving the stack in the condition shown in figure 5-3 (a) or (b). The program counter value saved on the stack is the address of the next instruction to be executed.
6. The T (Trace) bit of the status register is cleared to “0,” and the priority level of the interrupt is copied to bits I2 to I0, thus masking further interrupts unless they have a higher priority level. When an NMI is accepted, the interrupt mask level in bits I2 to I0 is set to 7.
7. The interrupt controller generates the vector address of the interrupt, and the entry at this address in the exception vector table is read to obtain the starting address of the user-coded interrupt handling routine.

In step 7, the same difference between the minimum and maximum modes exists as in the reset handling sequence. In the minimum mode, one word is copied from the vector table to the program counter, then the interrupt-handling routine starts executing from the address indicated in the program counter. In the maximum mode, two words are read. The lower byte of the first word is copied to the code page register. The second word is copied to the program counter. The interrupt-handling routine starts executing from the address indicated in the code page register and program counter.

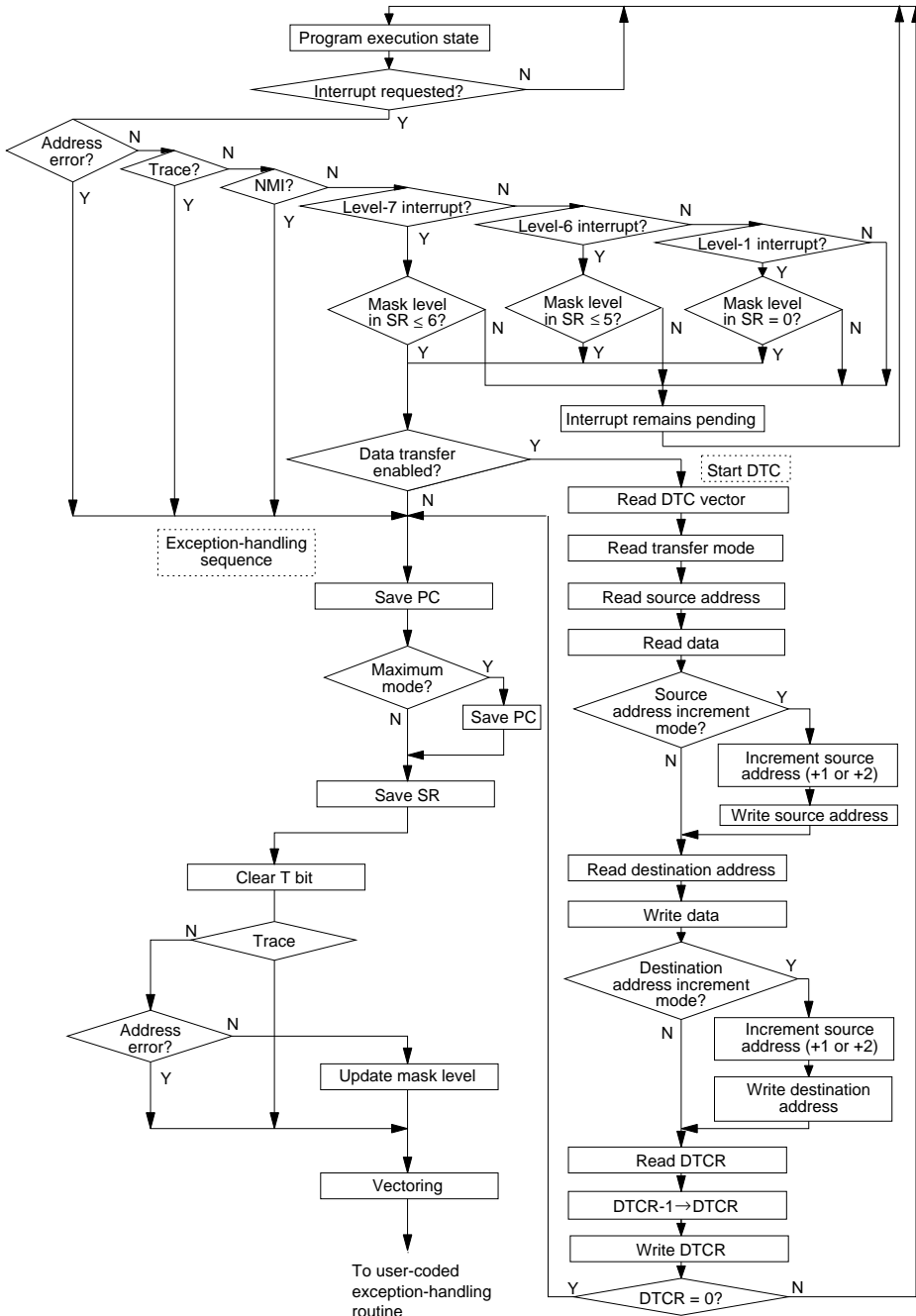


Figure 5-2 Interrupt Handling Flowchart

5.4.2 Stack Status after Interrupt Handling Sequence

Figure 5-3 (a) and (b) show the stack before and after the interrupt exception-handling sequence.

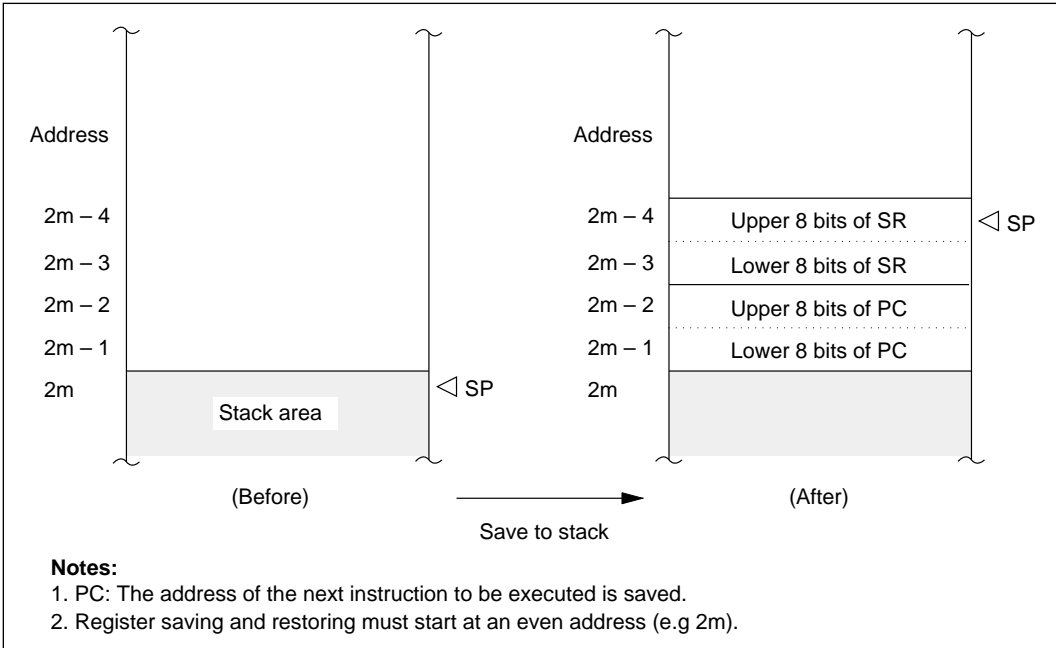


Figure 5-3 (a) Stack before and after Interrupt Exception-Handling (Minimum Mode)

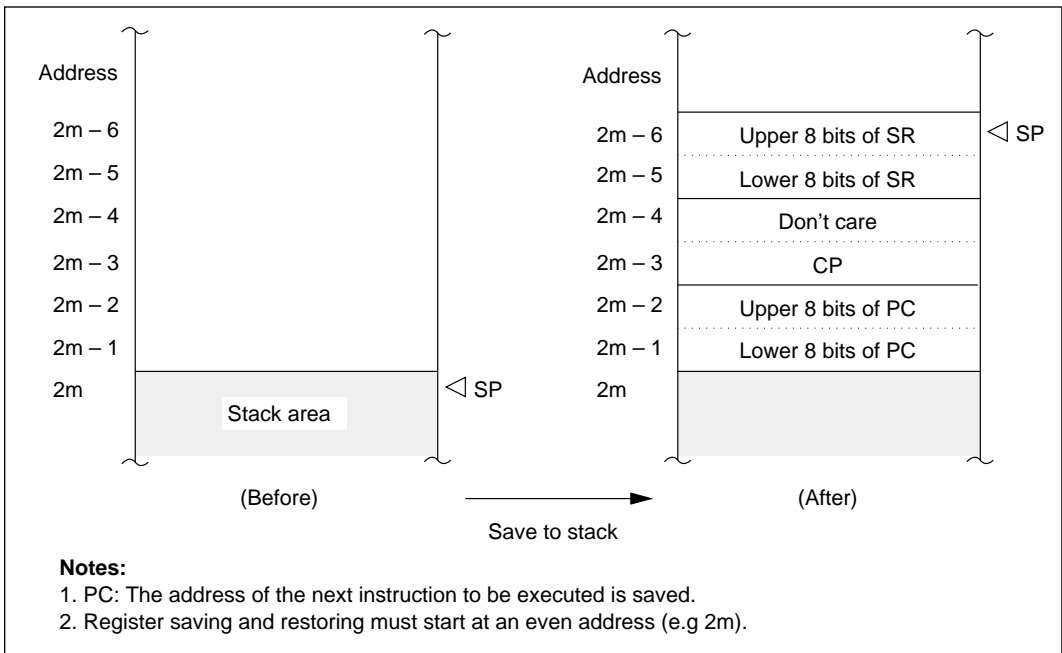


Figure 5-3 (b) Stack before and after Interrupt Exception-Handling (Maximum Mode)

5.4.3 Timing of Interrupt Exception-Handling Sequence

Figure 5-4 shows the timing of the exception-handling sequence for an interrupt when the program area and stack area are both in on-chip memory and the user-coded interrupt handling routine starts at an even address.

5.5 Interrupts During Operation of the Data Transfer Controller

If an interrupt is requested during a DTC data transfer cycle, the interrupt is not accepted until the data transfer cycle has been completed and the next instruction has been executed. This is true even if the interrupt is an NMI. An example is shown below.

(Example)

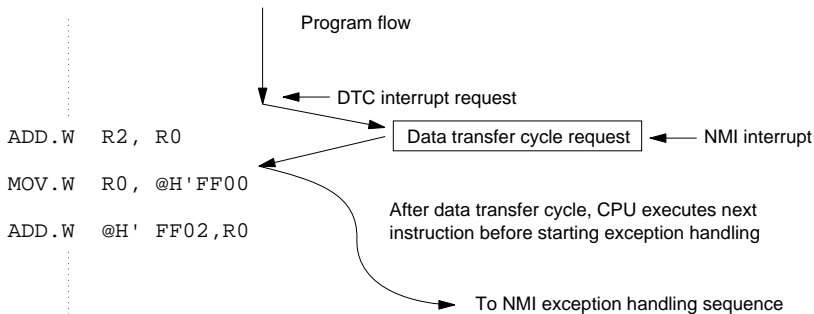
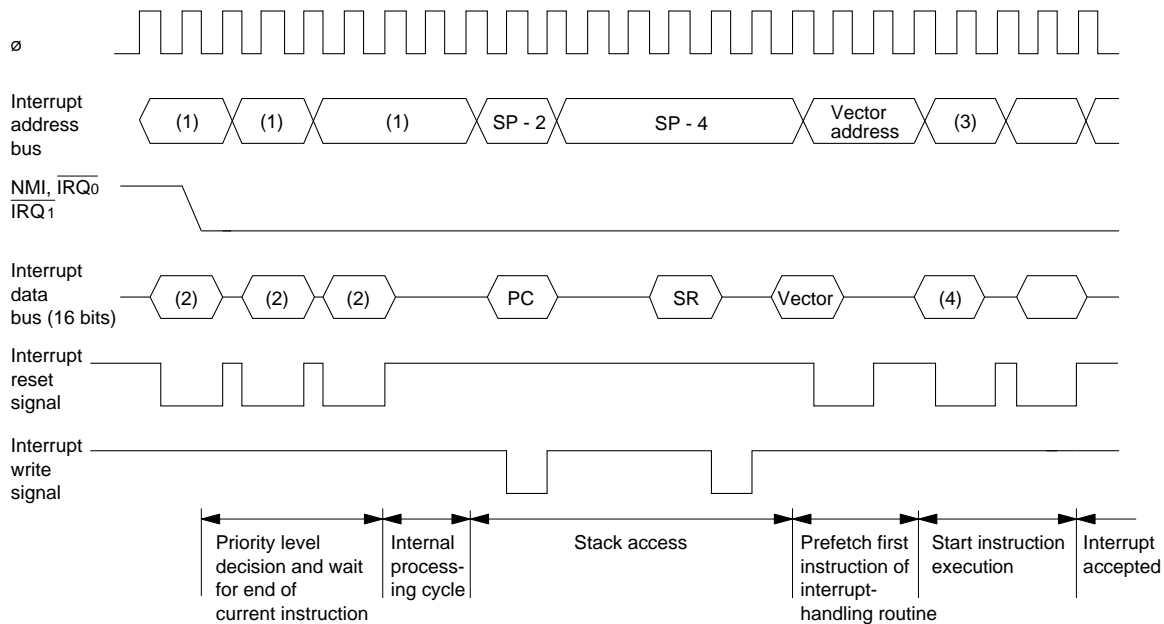


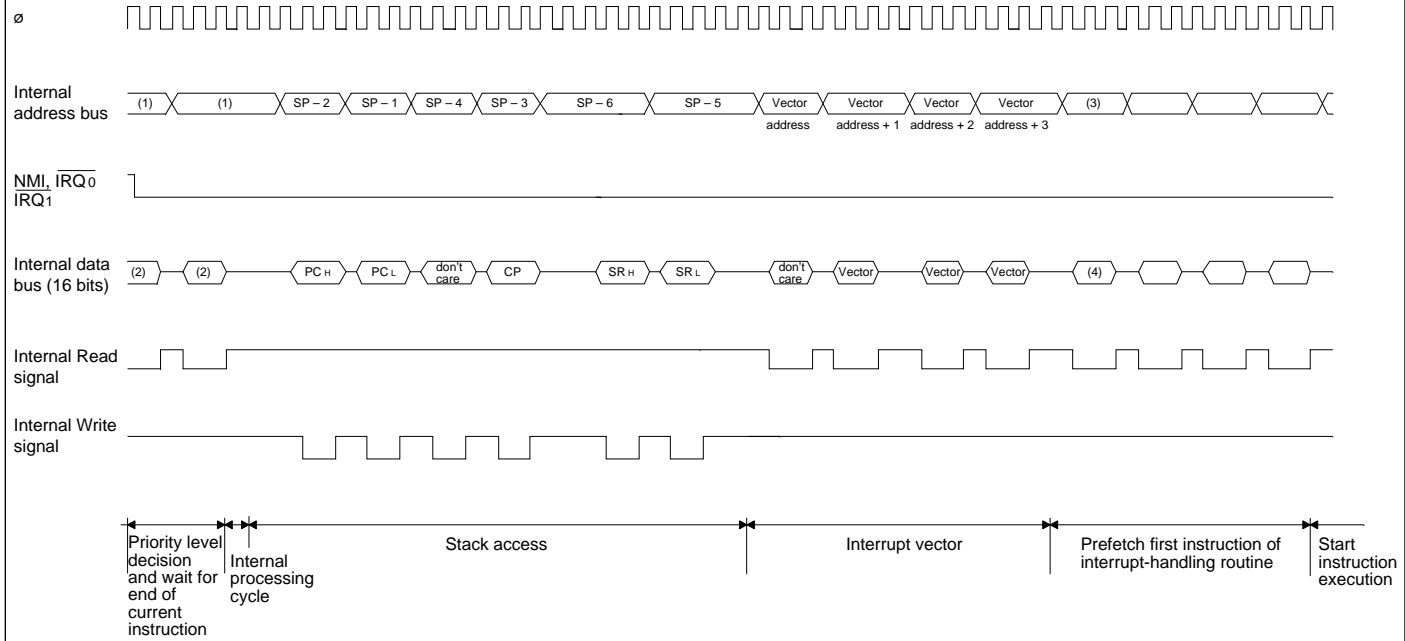
Figure 5-4 Interrupt Sequence (Minimum Mode, On-Chip Memory)



(1) Instruction prefetch address (2) Instruction code (3) Starting address of interrupt-handling routine (4) First instruction of interrupt-handling routine

Note: This timing chart applies to the minimum mode when the program and stack areas are both in on-chip memory and the interrupt-handling routine starts at an even address.

Figure 5-5 Interrupt Sequence (Maximum Mode, External Memory)



- (1) Instruction prefetch address
- (2) Instruction code
- (3) Starting address of interrupt-handling routine
- (4) First instruction of interrupt-handling routine

Note: This timing chart applies to the maximum mode when the program and stack areas are both in external memory. Instruction execution starts after interrupt vector fetch and 4-byte (4 bys cycles) instruction prefetch has been done.

5.6 Interrupt Response Time

Table 5-4 indicates the number of states that may elapse between the generation of an interrupt request and the execution of the first instruction of the interrupt-handling routine, assuming that the interrupt is not masked and not preempted by a higher-priority interrupt. Since word access is performed to on-chip memory areas, fastest interrupt service can be obtained by placing the program in on-chip ROM and the stack in on-chip RAM.

Table 5-4 Number of States before Interrupt Service

| No. | Reason for Wait | Number of States | |
|-------|---|-----------------------------------|--|
| | | Minimum Mode | Maximum Mode |
| 1 | Interrupt priority decision and comparison with mask level in CPU status register | 2 states | |
| 2 | Maximum number of states to completion of current instruction | Instruction is in on-chip memory | x (x = 38 for LDM instruction specifying all registers) |
| | | Instruction is in external memory | y (y = 74 + 16m for LDM instruction specifying all registers) |
| 3 | Saving of PC and SR or PC, CP, and SR and instruction prefetch | Stack is in on-chip RAM | 16 |
| | | Stack is in external memory | 28 + 6m |
| | Stack is in on-chip RAM | Instruction is in on-chip memory | 18 + x (56) |
| | | Instruction is in external memory | 23 + x (61) |
| Total | Stack is in external RAM | Instruction is in on-chip memory | 18 + y (92 + 16m) |
| | | Instruction is in external memory | 23 + y (97 + 16m) |
| | Stack is in external RAM | Instruction is in on-chip memory | 30 + 6m + x (68 + 6m) |
| | | Instruction is in external memory | 43 + 10m + x (81 + 10m) |
| | | Instruction is in external memory | 30 + 6m + y (104 + 22m) |
| | | | 43 + 10m + y (117 + 26m) |

Note: m: Number of wait states inserted in external memory access.
Values in parentheses are for the LDM instruction.