

# Section 4 Exception Handling

## 4.1 Overview

### 4.1.1 Types of Exception Handling and Their Priority

As indicated in table 4-1 (a) and (b), exception handling can be initiated by a reset, address error, trace, interrupt, or instruction. An instruction initiates exception handling if the instruction is an invalid instruction, a trap instruction, or a DIVXU instruction with zero divisor. Exception handling begins with a hardware exception-handling sequence which prepares for the execution of a user-coded software exception-handling routine.

There is a priority order among the different types of exceptions, as shown in table 4-1 (a). If two or more exceptions occur simultaneously, they are handled in their order of priority. An instruction exception cannot occur simultaneously with other types of exceptions.

**Table 4-1 (a) Exceptions and Their Priority**

	<b>Exception Type</b>	<b>Source</b>	<b>Detection Timing</b>	<b>Start of Exception-Handling Sequence</b>
High	Reset	External	$\overline{\text{RES}}$ Low-to-High transition	Immediately
	Address error	Internal	Instruction fetch or data read/write bus cycle	End of instruction execution
	Trace	Internal	End of instruction execution, if T = "1" in status register	End of instruction execution
	Interrupt	External, internal	End of instruction execution or end of exception-handling sequence	End of instruction execution
Low				

**Table 4-1 (b) Instruction Exceptions**

<b>Exception Type</b>	<b>Start of Exception-Handling Sequence</b>
Invalid instruction	Attempted execution of instruction with undefined code
Trap instruction	Started by execution of trap instruction
Zero divide	Attempted execution of DIVXU instruction with zero divisor

## 4.1.2 Hardware Exception-Handling Sequence

The hardware exception-handling sequence varies depending on the type of exception. When exception handling is initiated by a factor other than a reset, the CPU:

1. Saves the program counter and status register (in minimum mode) or program counter, code page register, and status register (in maximum mode) to the stack.
2. Clears the T bit in the status register to “0.”
3. Fetches the start address of the exception-handling routine from the exception vector table.
4. Branches to that address.

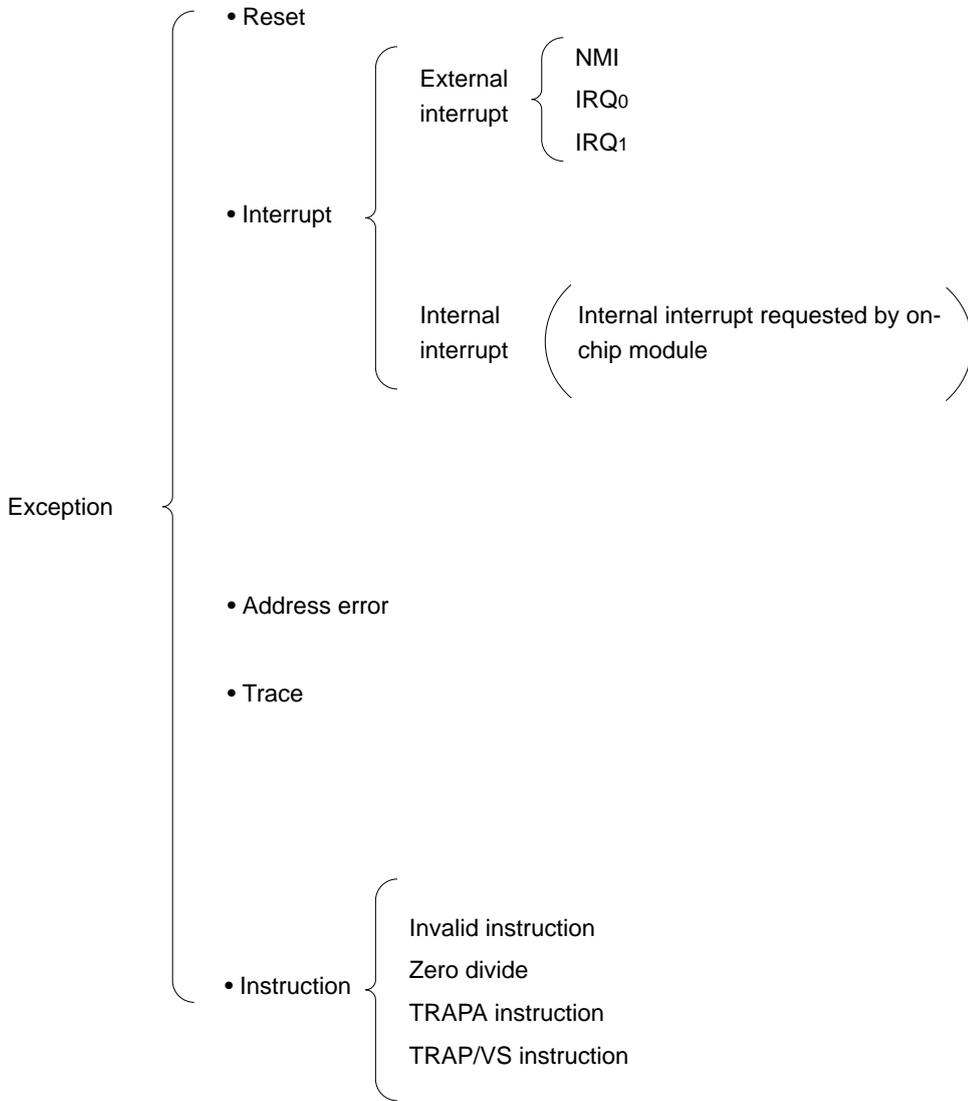
For an interrupt, the CPU also alters the interrupt mask level in bits I2 to I0 of the status register.

For a reset, step 1 is omitted. See section 4.2, “Reset,” for the full reset sequence.

## 4.1.3 Exception Factors and Vector Table

The factors that initiate exception handling can be classified as shown in figure 4-1.

The starting addresses of the exception-handling routines for each factor are contained in an exception vector table located in the low addresses of page 0. The vector addresses are listed in table 4-2. Note that there are different addresses for the minimum and maximum modes.



**Figure 4-1 Types of Factors Causing Exception Handling**

**Table 4-2 Exception Vector Table**

<b>Type of Exception</b>	<b>Vector Address</b>	
	<b>Minimum Mode</b>	<b>Maximum Mode <sup>*1</sup></b>
Reset (initialize PC)	H'0000 to H'0001	H'0000 to H'0003
— (Reserved for system)	H'0002 to H'0003	H'0004 to H'0007
Invalid instruction	H'0004 to H'0005	H'0008 to H'000B
DIVXU instruction (zero divide)	H'0006 to H'0007	H'000C to H'000F
TRAP/VS instruction	H'0008 to H'0009	H'0010 to H'0013
— (Reserved for system)	H'000A to H'000B to H'000E to H'000F	H'0014 to H'0017 to H'001C to H'001F
Address error	H'0010 to H'0011	H'0020 to H'0023
Trace	H'0012 to H'0013	H'0024 to H'0027
— (Reserved for system)	H'0014 to H'0015	H'0028 to H'002B
Nonmaskable external interrupt (NMI)	H'0016 to H'0017	H'002C to H'002F
— (Reserved for system)	H'0018 to H'0019 to H'001E to H'001F	H'0030 to H'0033 to H'003C to H'003F
TRAPA instruction (16 vectors)	H'0020 to H'0021 to H'003E to H'003F	H'0040 to H'0043 to H'007C to H'007F
External interrupts	IRQ <sub>0</sub> IRQ <sub>1</sub>	H'0040 to H'0041 H'0042 to H'0043
Internal interrupts <sup>*2</sup>	H'0044 to H'0045 to H'007E to H'007F	H'0088 to H'008B to H'00FC to H'00FF

**Notes:** \* 1. The exception vector table is located at the beginning of page 0.

\* 2. For details of the internal interrupt vectors, see table 5-2.

## 4.2 Reset

### 4.2.1 Overview

A reset has the highest exception-handling priority.

When the  $\overline{\text{RES}}$  pin goes Low, all current processing is halted and the H8/532 chip enters the reset state.

A reset initializes the internal status of the CPU and the registers of the on-chip supporting modules and I/O ports. It does not initialize the on-chip RAM.

When the  $\overline{\text{RES}}$  pin returns from Low to High, the H8/532 chip comes out of the reset state and begins executing the hardware reset sequence.

### 4.2.2 Reset Sequence

The Reset signal is detected when the  $\overline{\text{RES}}$  pin goes Low.

To ensure that the H8/532 is reset, the  $\overline{\text{RES}}$  pin should be held Low for at least 20ms at power-up. To reset the H8/532 during operation, the  $\overline{\text{RES}}$  pin should be held Low for at least 6  $\phi$  clock cycles. See table D-1, “Status of Ports” in Appendix D for the status of other pins in the reset state.

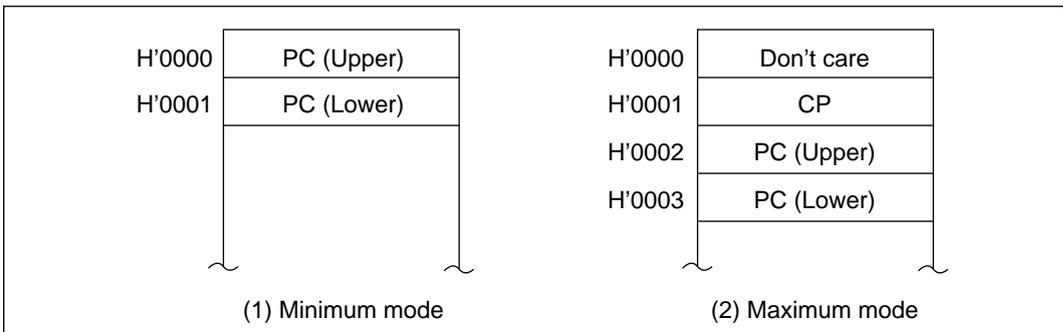
When the  $\overline{\text{RES}}$  pin returns to the High state after being held Low for the necessary time, the hardware reset exception-handling sequence begins, during which:

1. The value at the mode pins (MD2 to MD0) is latched in bits MDS2 to MDS0 of the mode control register (MDCR).
2. In the status register (SR), the T bit is cleared to disable the trace mode, and the interrupt mask level (bits I2 to I0) is set to 7. A reset disables all interrupts, including NMI.
3. The CPU loads the reset start address from the vector table into the program counter and begins executing the program at that address.

The contents of the vector table differs between minimum mode and maximum mode as indicated in figure 4-2. This affects step 3 as follows:

**Minimum mode:** One word is copied from addresses H'0000 and H'0001 in the vector table to the program counter. Program execution then begins from the address in the program counter (PC).

**Maximum Mode:** Two words are read from addresses H'0000 to H'0003 in the vector table. The byte in address H'0000 is ignored. The byte in address H'0001 is copied to the code page register (CP). The contents of addresses H'0002 and H'0003 are copied to the program counter. Program execution starts from the address indicated by the code page register and program counter.



**Figure 4-2 Reset Vector**

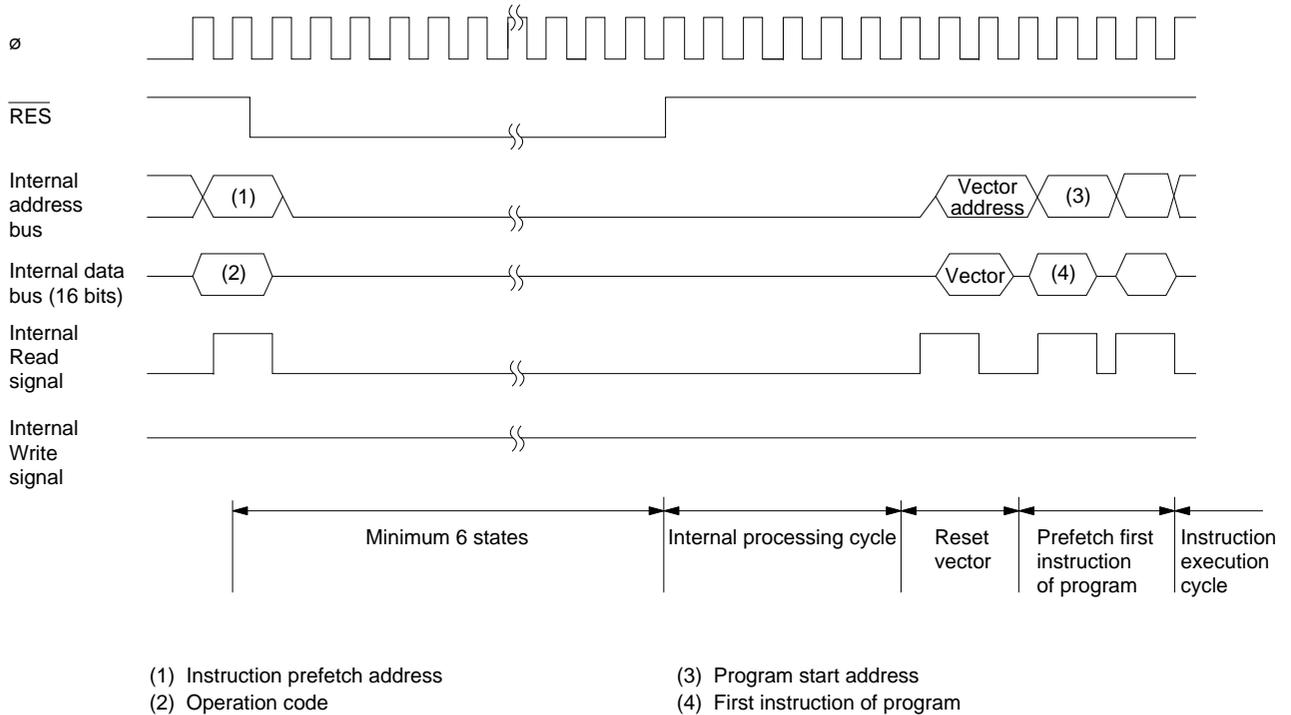
Figure 4-3 shows the timing of the reset sequence in minimum mode. Figure 4-4 shows the timing of the reset sequence in maximum mode.

### 4.2.3 Stack Pointer Initialization

The hardware reset sequence does not initialize the stack pointer, so this must be done by software. If an interrupt were to be accepted after a reset and before the stack pointer (SP) is initialized, the program counter and status register would not be saved correctly, causing a program crash. This danger can be avoided by coding the reset routine as explained next.

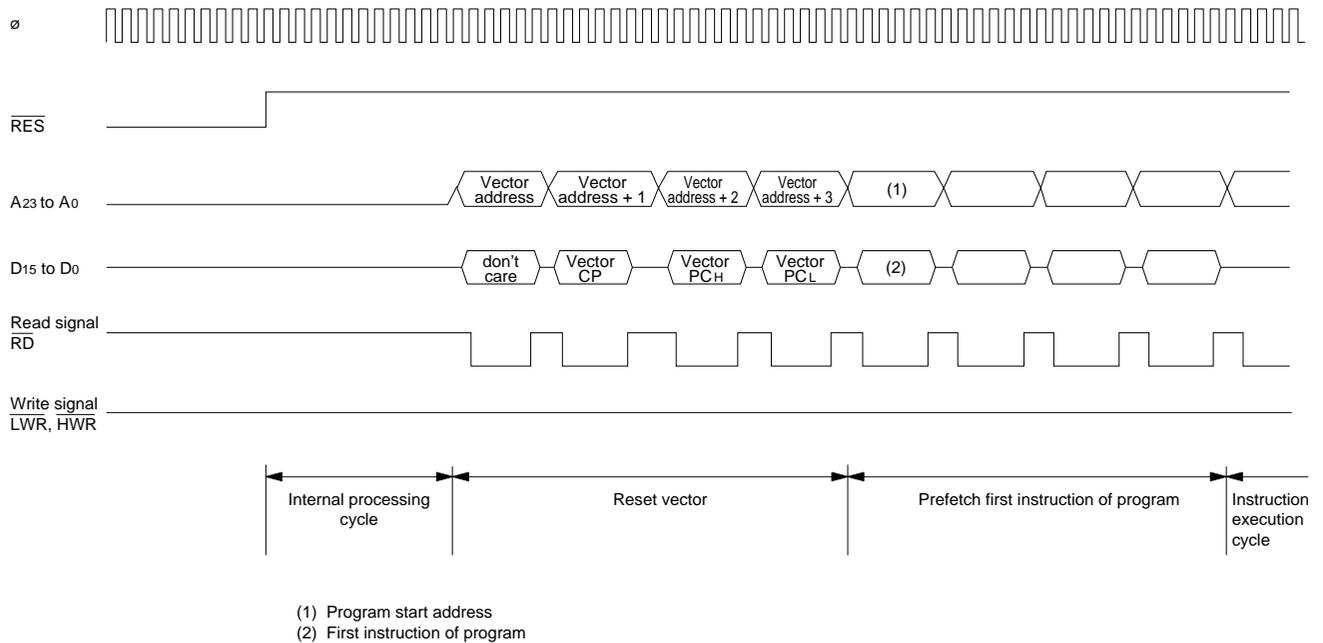
When the chip comes out of the reset state all interrupts, including NMI, are disabled, so the instruction at the reset start address is always executed. In the minimum mode, this instruction should initialize the stack pointer (SP). In the maximum mode, this instruction should be an LDC instruction initializing the stack page register (TP), and the next instruction should initialize the stack pointer. Execution of the LDC instruction disables interrupts again, ensuring that the stack pointer initializing instruction is executed.

Figure 4-3 Reset Sequence (Minimum Mode, On-Chip Memory)



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

Figure 4-4 Reset Sequence (Maximum Mode, External Memory)



**Note:** This diagram applies to maximum mode when the program area and vector table are both in external memory. After a reset, the wait-state controller inserts three wait states in each bus cycle.

## 4.3 Address Error

There are three causes of address errors:

- Illegal instruction prefetch
- Word data access at odd address
- Off-chip access in single-chip mode

An address error initiates the address error exception-handling sequence. This sequence clears the T bit of the status register to “0” to disable the trace mode, but does not affect the interrupt mask level in bits I2 to I0.

### 4.3.1 Illegal Instruction Prefetch

An attempt to prefetch an instruction from the register field in memory addresses H'FF80 to H'FFFF causes an address error regardless of the MCU operating mode.

Handling of this address error begins when the prefetch cycle that caused the error has been completed and execution of the current instruction has also been completed. The program counter value pushed on the stack is the address of the instruction immediately following the last instruction executed.

Program code should not be located in addresses H'FF7D to H'FF7F. If the CPU executes an instruction in these addresses, it will attempt to prefetch the next instruction from the register field, causing an address error.

### 4.3.2 Word Data Access at Odd Address

If an attempt is made to access word data starting at an odd address, an address error occurs regardless of the MCU operating mode. The program counter value pushed on the stack in the handling of this error is the address of the next instruction (or next but one) after the instruction that attempted the illegal word access.

### 4.3.3 Off-Chip Address Access in Single-Chip Mode

In the single-chip mode there is no external memory, so in addition to the address errors described above, the following two types of address errors can occur.

**Access to Addresses H'8000 to H'FB7F:** These addresses exist neither in on-chip ROM or RAM nor in the on-chip register field, so an address error occurs if they are accessed for any purpose: for instruction prefetch, byte data access, or word data access.

**Access to Disabled RAM Area:** The on-chip RAM area (H'FB80 to H'FF7F) can be disabled by clearing the RAME bit in the RAM control register (RAMCR). If RAM access is attempted in this state in the single-chip mode, an address error occurs.

## 4.4 Trace

When the T bit of the status register is set to “1,” the CPU operates in trace mode. A trace exception occurs at the completion of each instruction. The trace mode can be used to execute a program for debugging by a debugger.

In the trace exception sequence the T bit of the status register is cleared to “0” to disable the trace mode while the trace routine is executing. The interrupt mask level in bits I2 to I0 is not changed. Interrupts are accepted as usual during the trace routine.

In the status-register data saved on the stack, the T bit is set to “1.” When the trace routine returns with the RTE instruction, the status register is popped from the stack and the trace mode resumes.

If an address error occurs during execution of the first instruction after the return from the trace routine, since the address error has higher priority, the address error exception-handling sequence is initiated, clearing the T bit in the status register to “0” and making it impossible to trace this instruction.

## 4.5 Interrupts

Interrupts can be requested from three external sources (NMI, IRQ0, and IRQ1) and seven on-chip supporting modules: the 16-bit free-running timers (FRT1 to FRT3), the 8-bit timer, the serial communication interface (SCI), the A/D converter, and the watchdog timer (WDT). The on-chip interrupt sources can request a total of nineteen different types of interrupts, each having its own interrupt vector. Figure 4-5 lists the interrupt sources and the number of different interrupts from each source.

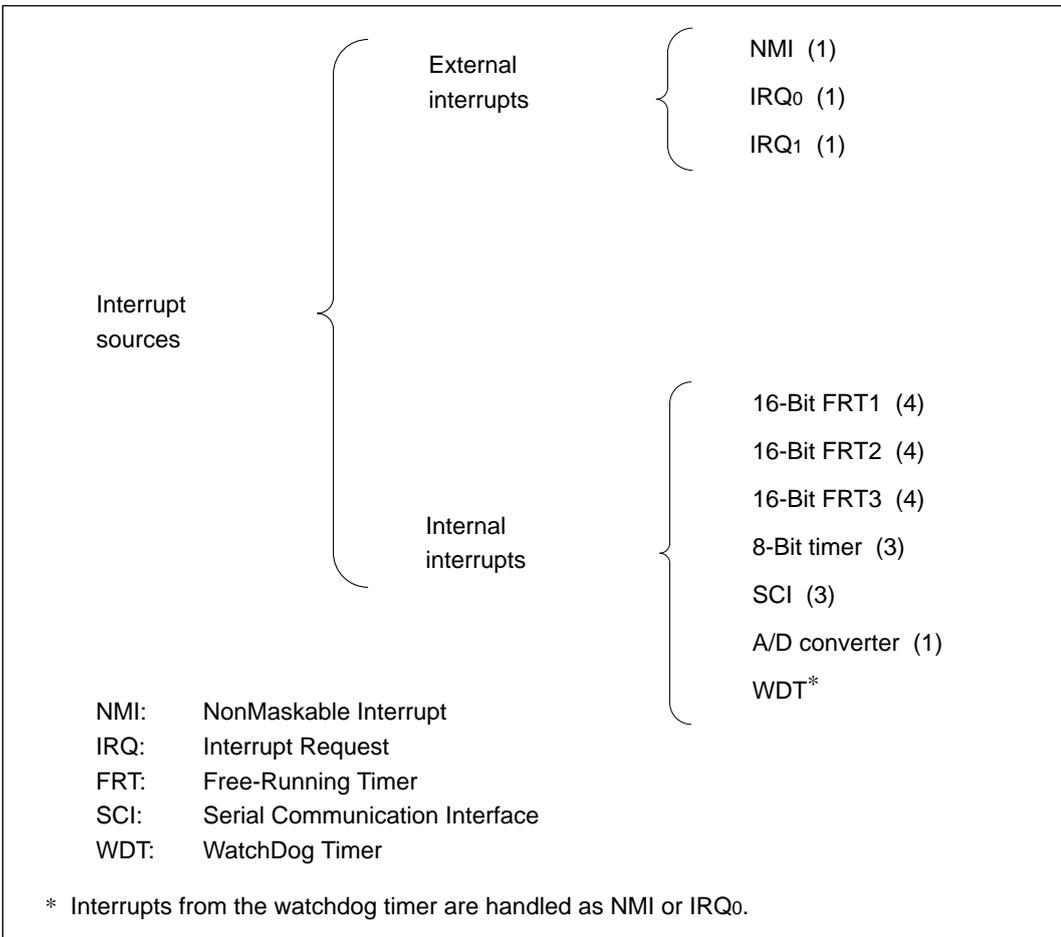
Each interrupt source has a priority. NMI interrupts have the highest priority, and are normally accepted unconditionally. The priorities of the other interrupt sources are set in control registers (IPR A to D) in the register field at the high end of page 0 and can be changed by software. Priority levels range from 0 (low) to 7 (high), with NMI considered to be on level 8.

The on-chip interrupt controller decides whether an interrupt can be accepted by comparing its priority with the interrupt mask level, and determines the order in which to accept competing interrupt requests. Interrupts that are not accepted immediately remain pending until they can be accepted later.

When it accepts an interrupt, the interrupt controller also decides whether to interrupt the CPU or start the on-chip data transfer controller (DTC). This decision is controlled by bits set in four data transfer enable registers (DTE A to D) in the register field. The DTC is started if the corresponding DTE bit is set to “1;” otherwise a CPU interrupt is generated. DTC interrupts provide an efficient way to send and receive blocks of data via the serial communication interface, or to transfer data between memory and I/O without detailed CPU programming. The CPU stops while the DTC is operating. DTC interrupts are described in section 6, “Data Transfer Controller.”

The hardware exception-handling sequence for a CPU interrupt clears the T bit in the status register to “0” and sets the interrupt mask level in bits I2 to I0 to the level of the interrupt it has accepted. This prevents the interrupt-handling routine from being interrupted except by a higher-level interrupt. The previous interrupt mask level is restored on the return from the interrupt-handling routine.

For further information on interrupts, see section 5, “Interrupt Controller.”



**Figure 4-5 Interrupt Sources (and Number of Interrupt Types)**

## 4.6 Invalid Instruction

An invalid instruction exception occurs if an attempt is made to execute an instruction with an undefined operation code or illegal addressing mode specification. The program counter value pushed on the stack is the value of the program counter when the invalid instruction code was detected.

In the invalid instruction exception-handling sequence the T bit of the status register is cleared to “0,” but the interrupt mask level (I2 to I0) is not affected.

## 4.7 Trap Instructions and Zero Divide

A trap exception occurs when the TRAPA or TRAP/VS instruction is executed. A zero divide exception occurs if an attempt is made to execute a DIVXU instruction with a zero divisor.

In the exception-handling sequences for these exceptions the T bit of the status register is cleared to “0,” but the interrupt mask level (I2 to I0) is not affected. If a normal interrupt is requested while a trap or zero-divide instruction is being executed, after the trap or zero-divide exception-handling sequence, the normal interrupt exception-handling sequence is carried out.

**TRAPA Instruction:** The TRAPA instruction always causes a trap exception. The TRAPA instruction includes a vector number from 0 to 15, allowing the user to provide up to sixteen different trap-handling routines.

**TRAP/VS Instruction:** When the TRAP/VS instruction is executed, a trap exception occurs if the overflow (V) bit in the condition code register is set to “1.” If the V bit is cleared to “0,” no exception occurs and the next instruction is executed.

**DIVXU Instruction with Zero Divisor:** An exception occurs if an attempt is made to divide by zero in a DIVXU instruction.

## 4.8 Cases in Which Exception Handling is Deferred

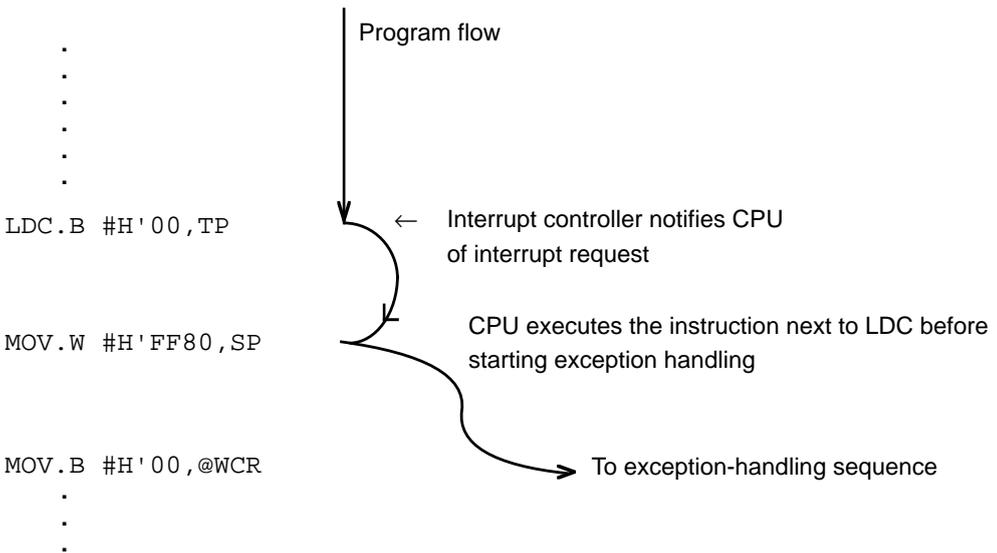
In the cases described next, the address error exception, trace exception, external interrupt (NMI, IRQ0, and IRQ1) requests, and internal interrupt requests (19 types) are not accepted immediately but are deferred until after the next instruction has been executed.

### 4.8.1 Instructions that Disable Interrupts

Interrupts are disabled immediately after the execution of five instructions: XORC, ORC, ANDC, LDC, and RTE.

Suppose that an internal interrupt is requested and the interrupt controller, after checking the interrupt priority and interrupt mask level, notifies the CPU of the interrupt, but the CPU is

currently executing one of the five instructions listed above. After executing this instruction the CPU always proceeds to the next instruction. (And if the next instruction is one of these five, the CPU also proceeds to the next instruction after that.) The exception-handling sequence starts after the next instruction that is not one of these five has been executed. The following is an example: (Example)



#### 4.8.2 Disabling of Exceptions Immediately after a Reset

If an interrupt is accepted after a reset and before the stack pointer (SP) is initialized, the program counter and status register will not be saved correctly, leading to a program crash. To prevent this, when the chip comes out of the reset state all interrupts, including the NMI, are disabled, so the first instruction of the reset routine is always executed. As noted earlier, in the minimum mode, this instruction should initialize the stack pointer (SP). In the maximum mode, the first instruction should be an LDC instruction that initializes the stack page register (TP); the next instruction should initialize the stack pointer.

#### 4.8.3 Disabling of Interrupts after a Data Transfer Cycle

If an interrupt starts the data transfer controller and another interrupt is requested during the data transfer cycle, when the data transfer cycle ends, the CPU always executes the next instruction before handling the second interrupt.

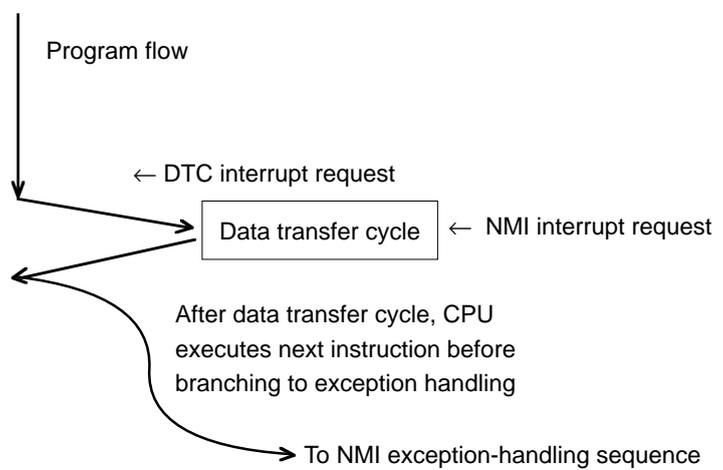
Even if a nonmaskable interrupt (NMI) occurs during a data transfer cycle, it is not accepted until the next instruction has been executed. An example of this is shown below.

(Example)

```

.
.
.
.
.
ADD.W R2, R0
MOV.W R0, @H' FF00
MOV.W #H' FF02, R0
.
.
.

```



### 4.9 Stack Status after Completion of Exception Handling

The status of the stack after an exception-handling sequence is described below.

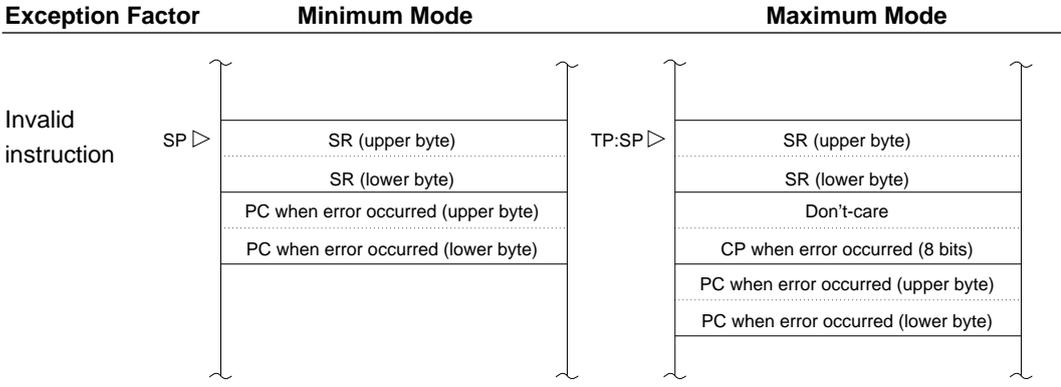
Table 4-3 shows the stack after completion of the exception-handling sequence for various types of exceptions in the minimum and maximum modes.

**Table 4-3 Stack after Exception Handling Sequence**

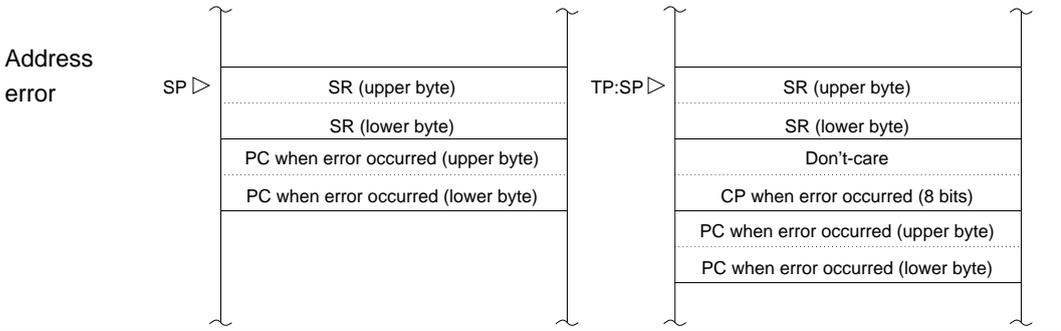
Exception Factor	Minimum Mode	Maximum Mode
Trace	SP ▷ [ SR (upper byte) ]	TP:SP ▷ [ SR (upper byte) ]
Interrupt	[ SR (lower byte) ]	[ SR (lower byte) ]
	[ Next instruction address (upper byte) ]	[ Don't-care ]
Trap	[ Next instruction address (lower byte) ]	[ Next instruction page (8 bits) ]
		[ Next instruction address (upper byte) ]
Zero divide (DIVXU)		[ Next instruction address (lower byte) ]

**Note:** The RTE instruction returns to the next instruction after the instruction being executed when the exception occurred.

**Table 4-3 Stack after Exception Handling Sequence (cont)**



**Note:** The program counter value pushed on the stack is not necessarily the address of the first byte of the invalid instruction.



**Note:** The program counter value pushed on the stack is the address of the next instruction after the last instruction successfully executed.

#### **4.9.1 PC Value Pushed on Stack for Trace, Interrupts, Trap Instructions, and Zero Divide Exceptions**

The program counter value pushed on the stack for a trace, interrupt, trap, or zero divide exception is the address of the next instruction at the time when the interrupt was accepted. The RTE instruction accordingly returns to the next instruction after the instruction executed before the exception-handling sequence.

#### **4.9.2 PC Value Pushed on Stack for Address Error and Invalid Instruction Exceptions**

The program counter value pushed on the stack for an address error or invalid instruction exception differs depending on the conditions when the exception occurred.

### **4.10 Notes on Use of the Stack**

If the stack pointer is set to an odd address, an address error will occur when the stack is accessed during interrupt handling or for a subroutine call. The stack pointer should always point to an even address. To keep the stack pointer pointing to an even address, a program should use word data size when saving or restoring registers to and from the stack.

In the @-SP or @SP+ addressing mode, the CPU performs word access even if the instruction specifies byte size. (This is not true in the @-Rn and @Rn+ addressing modes when Rn is a register from R0 to R6.)