Faults

6

CHAPTER 6
FAULTS

This chapter describes the fault handling facilities of the 80960SA/SB processor. The subjects covered include the fault-handling data structures, the software support required for fault handling, and the fault handling mechanism. A reference section that contains detailed information on each fault type is provided at the end of the chapter.

OVERVIEW OF THE FAULT-HANDLING FACILITIES

The processor is able to detect various conditions in code or in its internal state (called *fault conditions*) that could cause the processor to deliver incorrect or inappropriate results or that could cause it to head down an undesirable control path. For example, the processor recognizes divide-by-zero and overflow conditions on integer calculations. It also detects inappropriate operand values, uncompleted memory accesses, or references to incomplete or non-existent architecture-defined data structures.

The processor can detect a fault while it is executing a program, an interrupt handler, or a fault handler. (In this chapter, when a program is referred to, it generally also means any interrupt handler or fault handler that may have been invoked while the processor was working on the program.)

When the processor detects a fault, it handles the fault immediately and independently of the program or handler it is currently working on, using a mechanism similar to that used to service interrupts.

A fault is generally handled with a fault-handling procedure (called a fault handler), which the processor invokes through an implicit procedure call. Prior to making the call, the processor saves the state of the current program and in some cases the state of an incomplete instruction. It also saves information about the fault, which the fault handler can use to correct or recover from the condition that caused the fault.

If the fault handler is able to recover from the fault, the processor can then restore the program to its state prior to the fault and resume work on the program. If the fault handler is not able to recover from the fault, it can take any of several actions to gracefully shut down the processor.

FAULT TYPES

All of the faults that the processor detects are predefined. These faults are divided into types and subtypes, each of which is given a number. When the processor detects a fault, it records the fault type and subtype in a fault record. It then uses the type number to select a fault handler. The fault handler has the option of using the subtype number to select a specific fault-handling procedure.

Table 6-1 lists the faults that the processor detects, arranged by type and subtype. For convenience, individual faults are referred to in this manual by their fault-subtype name. Thus an *arithmetic-integer-overflow fault* is referred to as an *integer-overflow fault.*

The fifth column of Table 6-1 shows each fault as it appears in the fault record (the word at offset 40 of the fault record is shown later in this chapter).

Table 6-1: Fault Types and Subtypes

Fault Type		**Fault Subtype**		**Fault Record**
No.	**Name**	**No./Bit Position**	**Name**	
1	Trace	Bit 1	Instruction Trace	$XX01\ XX02_{16}$
		Bit 2	Branch Trace	$XX01\ XX04_{16}$
		Bit 3	Call Trace	$XX01\ XX08_{16}$
		Bit 4	Return Trace	$XX01\ XX10_{16}$
		Bit 5	Prereturn Trace	$XX01\ XX20_{16}$
		Bit 6	Supervisor Trace	$XX01\ XX40_{16}$
		Bit 7	Breakpoint Trace	$XX01\ XX80_{16}$
2	Operation	1	Invalid Opcode	$XX02\ XX01_{16}$
		4	Invalid Operand	$XX02\ XX04_{16}$
3	Arithmetic	1	Integer Overflow	$XX03\ XX01_{16}$
		2	Arithmetic Zero-Divide	$XX03\ XX02_{16}$
4	Floating Point	Bit 0	Floating Overflow	$XX04\ XX01_{16}$
		Bit 1	Floating Underflow	$XX04\ XX02_{16}$
		Bit 2	Floating Invalid-Operation	$XX04\ XX04_{16}$
		Bit 3	Floating Zero-Divide	$XX04\ XX08_{16}$
		Bit 4	Floating Inexact	$XX04\ XX10_{16}$
		Bit 5	Floating Reserved-Encoding	$XX04\ XX20_{16}$
5	Constraint	1	Constraint Range	$XX05\ XX01_{16}$
		2	Privileged	$XX05\ XX02_{16}$
7	Protection	Bit 1	Length	$XX07\ XX02_{16}$
A	Type	1	Type Mismatch	$XX0A\ XX01_{16}$

NOTE

The i960 architecture defines a basic set of fault types and subtypes. Processors that provide extensions to the architecture may recognize additional fault conditions. The encoding of fault types and subtypes allows any of these extensions to be included in the fault table along with the basic faults. Space in the fault table will be reserved in such a way that processors that recognize the same fault types and subtypes will encode them in the same way.

For example, the floating-point faults (fault type 4) are an extension provided in the 80960SB processor (but not in the 80960SA processor). Any other processors based on the i960 architecture that also recognize floating-point faults will also encode them as fault type 4.

FAULT-HANDLING METHOD

The processor handles all faults through a procedure call to a fault handler. When a fault occurs while the processor is executing a program, the processor determines the fault type, then selects a fault handler for that type from an architecture-defined data structure called the *fault table*. It then invokes the fault handler (by means of an implicit call). As described later in this chapter, the fault-handler call can be a local call (call-extended operation), a local system-procedure-table call (local system-call operation), a supervisor call, or a trace-table call.

Before the processor begins executing the fault-handler procedure, it creates a fault record on its current stack (i.e., the stack being used by the fault handler). This record includes information on the state of the program and data on the fault. If the fault occurred while the processor was in the midst of executing an instruction, a resumption record for the instruction may also be saved on the stack.

Following the creation of the fault and resumption records, the processor begins executing the selected fault-handler procedure.

This same procedure call method is used to handle faults that occur while the processor is servicing an interrupt or that occur while the processor is working on another fault handler.

Multiple Fault Conditions

It is possible for multiple fault conditions to occur simultaneously. For certain fault types, such as trace faults or protection faults, bit positions in the fault-subtype field are used to indicate the occurrence of multiple faults of the same type. As a general rule, however, the processor does not indicate situations where multiple faults occur. Instead, it generates one of the faults and does not report on the faults that were not generated.

SOFTWARE REQUIREMENTS FOR HANDLING FAULTS

To use the processor's fault-handling facilities, the following data structures and procedures must be present in memory:

- Fault Table
- Trace Table
- System-Procedure Table (Optional)
- Fault-Handler Procedures

Software should generally load these items in memory as part of the initialization procedure. Once they are present in memory and pointers to them have been included in the IMI, the processor then handles faults automatically and independently from software.

Requirements for the fault table, trace table, and fault-handler procedures are given in the following sections.

FAULT TABLE

The fault table provides the processor with a pathway to the fault-handler procedures. As shown in Figure 6-1, there is one entry in the fault table for each fault type. When a fault occurs, the processor uses the fault type to select an entry in the fault table. From this entry, the processor then obtains a pointer to the fault-handler procedure for the type of fault that occurred.

Once a fault-handler procedure has been called, it has the option of reading the fault subtype or subtypes from the fault record to determine the appropriate fault recovery action.

Location of the Fault Table in Memory

The fault table can be located anywhere in the address space. The processor obtains a pointer to the fault table from the IMI.

Fault-Table Entries

As shown at the bottom of Figure 6-1, three types of fault-table entries are allowed: a local-procedure entry, a system-procedure-table entry, and a trace-table entry. Each entry type is two words long. The entry-type field (bits 0 and 1 of the first word of the entry) and the address in the second word of the entry determines the entry type.

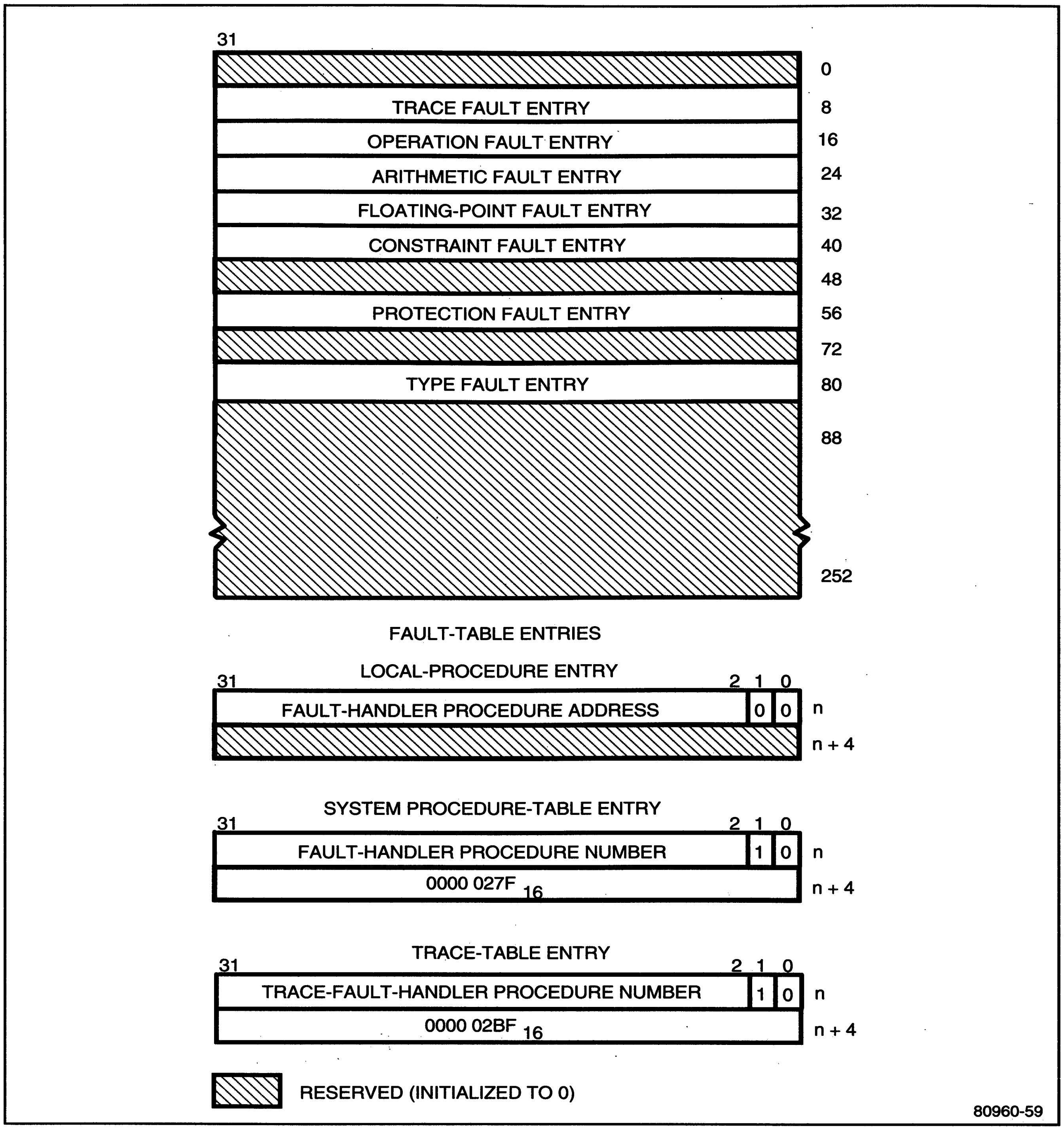

Figure 6-1: Fault Table and Fault-Table Entries

A local-procedure entry (entry type 00_2) provides an instruction pointer (address in the address space) for the fault-handler procedure. Using this entry, the processor invokes the specified fault handler by means of an implicit call-extended operation (similar to that performed for the **callx** instruction). The second word of a local-procedure entry is reserved. It should be set to zero when the fault table is created and not accessed after that.

A system-procedure-table entry provides a procedure number in the system-procedure table. This entry must have an entry type of 10_2 and an address in the second word of $0000027F_{16}$ (the address of the system-procedure table). Using this entry, the processor invokes the specified fault handler by means of an implicit call-system operation (similar to that performed for the **calls** instruction). Fault-handling procedures in the system-procedure table can be local procedures or supervisor procedures.

The trace table entry provides a procedure number in a special procedure table called the *trace table*. This entry must have an entry type of 10_2 and a value in the second word of $000002BF_{16}$. The function of this entry is described in the following section titled "Trace-Fault Handling".

To summarize, a fault handler can be invoked through the fault table in any of four ways: a local procedure call; a local system-procedure-table call; a supervisor call; or a trace table call.

TRACE-FAULT HANDLING

When handling trace faults, the i960 architecture requires that tracing be disabled (i.e., the trace-enable flag of the process controls must be set to 0). To support this requirement, the architecture defines a special trace table. This procedure table has the same structure as the system-procedure table shown in Figure 4-4, but with the following two restrictions:

- All entries must be supervisor entries (10_2 in bits 0 and 1).
- The trace control flag (byte 12, bit 0) must be set to 0.

The supervisor stack pointer in the trace table should be the same as the stack pointer given in the system-procedure table.

The effect of these restrictions is that on a call to a trace-fault handler routine, the processor saves the current state of the trace-enable flag and then clears the flag to disable tracing. On the return from the trace fault handler, the processor automatically restores the trace-enable flag to the state it was in prior to the trace fault.

The second word of the trace fault entry of the fault table must contain the value $000002FF_{16}$. The trace table generally will have only one procedure entry, which points to the trace-fault-handler procedure. However, this procedure table can be used as a pathway to other fault-handler routines.

This method of handling trace faults must always be used except for the following circumstances:

- If tracing is never going to be used (i.e., the trace-enable flag of the process controls is always set to 0), the trace table is not required.
- If tracing is never going to be used on supervisor calls, the system-procedure table can be used in place of the trace table, since the trace-control flag of the system-procedure table will then be set to 0.

In the latter case, the trace-fault handler must still be called with a supervisor call. Also, the value given in the second word of the trace-fault entry in the fault table, must be $0000027F_{16}$.

FAULT-HANDLER PROCEDURES

The fault-handler procedures can be located anywhere in the address space. Each procedure must begin on a word boundary.

The processor can execute the procedure in the user mode or the supervisor mode, depending on the type of fault table entry.

NOTE

> To resume work on a program at the point where a fault occurred (following the recovery action of the fault handler), the fault handler must be executed in the supervisor mode. The reason for this requirement is described in a following section titled "Returning with Resumption."

Possible Fault-Handler Actions

Many of the faults that occur can be recovered from easily. When recovery from the fault is possible, the processor's fault-handling mechanism allows the processor to automatically resume work on the program or interrupt that it was working on when the fault occurred. The resumption action is initiated with a **ret** instruction in the fault-handler procedure.

If recovery from the fault is not possible or not desirable, the fault handler can take one of the following actions, depending on the nature and severity of the fault condition (or conditions, in the case of multiple faults):

- Return to a point in the program or interrupt code other than the point of the fault
- Save the current state of the processor and call a debug monitor

When working with the processor at the development level, a common action of the fault handler is to save the fault and processor state information and make a call to a debugging device such as a debugging monitor. This device can then be used to analyze the fault.

Program and Instruction Resumption Following a Fault

Faults can occur prior to the execution of the faulting instruction (i.e., the instruction that causes the fault), during the instruction, or after the instruction. When the fault occurs before the faulting instruction is executed, the instruction can theoretically be executed on the return from the fault handler. So, the fault can be handled in such a way as to not interrupt in the control flow of the program.

When a fault occurs during the instruction that caused a fault, the fault may be accompanied by a change in the state of the program such that the execution of the program can not be resumed after the fault has been handled. For example, when an integer-overflow fault occurs, the overflow value is stored in the destination. If the destination register was the same as one of the source registers, the source value is lost, making it impossible to reexecute the faulting instruction.

In general, resumption of program execution with no changes in the program's control flow is always possible with the following fault types or subtypes:

- All Operation Subtypes
- Arithmetic Zero-Divide
- All Floating-Point Subtypes Except Floating Inexact
- All Constraint Subtypes
- All Trace Subtypes

Resumption of the program may or may not be possible with the following fault types and subtypes:

- Integer Overflow
- Floating Inexact

The effect that specific fault types have on a program is given in the fault reference section at the end of this chapter under the heading "Program State Changes."

Returning With Resumption

As described above, certain faults do not change the state of the program when they occur, even if the execution of the instruction was suspended as part of the fault-generation mechanism. Here, the processor allows work on a program to be resumed at the point where the fault occurred (including resumption of a suspended instruction), following a return from a fault handler. The resumption mechanism is similar to that provided for returning from an interrupt handler.

To use this mechanism, the fault handler must be invoked using a supervisor call. This method is required because to resume work on the program and a suspended instruction at the point where the fault occurred, the saved process controls in the fault record must be copied back into the processor on the return from the fault handler. The processor only performs this action if the processor is in the supervisor mode on the return.

If the fault handler is invoked with a local-procedure call or a local-procedure-table call, the return IP determines where in the program the processor resumes work, following a return from a fault handler. Here, the return is handled in a similar manner to a return from an explicit call with a **call** or **callx** instruction.

The return IP (referred to later in this chapter as the saved IP) is saved in the RIP register (r2) of the stack frame that was in use when the fault occurred. This IP may be the instruction the processor faulted on or the next instruction that the processor would have executed if the fault had not occurred. In either case, the resumption record is not used, so the processor might continue work on the program without completing the instruction that the fault occurred on.

A fault handler should thus be invoked with a local-procedure or local-procedure-table call only if it is not required or desirable to resume the program at the point where the fault occurred. The following section titled "Returning Without Resumption" discusses returning to a point in the program code other than the point of the fault.

Returning Without Resumption

There may be situations where the fault handler needs to return to a point in the program other than where the fault occurred. This can be done by altering the return IP in the previous frame. However, if resumption information was collected with the fault (resulting in the resume flag being set in the saved process controls), such a return can cause unpredictable results.

To predictably perform a return from a fault handler to an alternate point in the program, the fault handler should perform the following two steps:

1. Flush the local register sets to the stack with a **flushreg** instruction.
2. Clear the following information in the process-controls field of the fault record before the return: the resume and trace-fault-pending flags; the internal state field.

NOTE

This technique should be used carefully and only in situations where the fault handler is closely coupled with the application program. Also, a return of this type can only be performed if the processor is in supervisor mode prior to the return.

Aborting a Program

Where it is not possible to return to the program in which a fault occurred, the fault handler can be designed to abort the program. Several possible actions that a fault handler can take when aborting a program are given in the section earlier in this chapter titled "Possible Fault-Handler Actions."

FAULT CONTROLS

Certain fault types and subtypes have masks or flags associated with them that determine whether or not a fault is generated when a fault condition occurs. Table 6-2 lists these flags and masks, the data structures in which they are located, and the fault subtype they affect.

Table 6-2: Fault Flags or Masks

Flag or Mask Name	Location	Fault Affected
Integer Overflow Mask	Arithmetic Controls	Integer Overflow
Floating Overflow Mask	Arithmetic Controls	Floating Overflow
Floating Underflow Mask	Arithmetic Controls	Floating Underflow
Floating Invalid Operation Mask	Arithmetic Controls	Floating Invalid Operation
Floating Zero-Divide Mask	Arithmetic Controls	Floating Zero-Divide
Floating-point Inexact Mask	Arithmetic Controls	Floating Inexact
No Imprecise Faults Flag	Arithmetic Controls	All Imprecise Faults
Trace-Enable Flag	Process Controls	All Trace Faults
Trace-Mode Flags	Trace Controls	All Trace Faults

The integer and floating-point mask bits inhibit faults from being raised for specific fault conditions (i.e., integer overflow and floating-point overflow, underflow, zero divide, invalid operation, and inexact). The use of these masks is discussed in the fault-reference section at the end of this chapter. Also, the floating-point fault masks are described in Chapter 10 in the section titled "Exceptions and Fault Handling."

The no-imprecise-faults (NIF) flag controls the synchronizing of faults for a category of faults called imprecise faults. This flag should be set to 1. The function of this flag is described later in this chapter in the section titled "Precise and Imprecise Faults."

NOTE

In the 80960SA/SB implementation of the i960 architecture, the NIF flag is ignored. This implementation assumes that the flag is set to 1, meaning that all faults are required to be precise.

The trace-mode flags (in the trace controls) and trace-enable flag (in the process controls) support trace faults. The trace-mode flags enable trace modes; the trace-enable flag enables the generation of trace faults. The use of these flags is described in the fault reference section on trace faults at the end of this chapter. Further discussion of these flags is provided in Chapter 7 in the section titled "Trace-Enable and Trace-Fault-Pending Flags."

Faults and Interrupts

If an interrupt occurs during an instruction that will fault or that has just faulted, the processor will handle the fault in the following way

1. It completes selection of the fault handler, writing the fault record (and resumption record if needed) to the stack. The RIP of the fault handler points to the interrupted/faulted routine.
2. It selects the interrupt handler, writing the interrupt record to the stack. The RIP of the interrupt handler points to the first instruction of the fault record.
3. It completes the interrupt handler, returning to the fault handler.
4. It completes the fault handler, returning to the interrupted/faulted routine.

GENERATING A FAULT

The processor generates faults implicitly when fault conditions occur and explicitly at the request of software. Most faults are generated implicitly. The fault control bits described in the previous section allow the implicit generation of some faults to be either enabled (as with the trace faults) or masked (as with the floating-point faults).

Explicit Generation of Faults

Two sets of instructions allow faults to be generated explicitly anywhere within an application program, kernel procedure, interrupt handler, or fault handler. The fault-if instructions (**faulte**, **faultne**, **faultl**, **faultle**, **faultg**, **faultge**, **faulto**, and **faultno**) allow a conditional fault to be generated. When one of these instructions is executed, the processor checks the condition code bits in the arithmetic controls, then generates a constraint-range fault if the condition specified with the instruction is met.

The **mark** and force mark (**fmark**) instructions allow a breakpoint trace fault to be generated anywhere in the instruction stream.

FAULT RECORD

When a fault occurs, the processor records information about the fault in a fault record. This record is stored on the stack that the fault-handler procedure will use. (The location of the fault record on the stack is described later in this chapter in the section titled "Location of the Fault and Resumption Records.") The fault handler and processor use the information in the fault record to recover from or correct the fault condition and resume execution of the program. Figure 6-2 shows the structure of the fault record. The use of the fields in this record are described in the following paragraphs.

The type number (byte ordinal) of a fault is stored in the fault-type field; the subtype number or bit positions (byte ordinal) is stored in the fault-subtype field.

The fault-flags field provides a set of general-purpose flags that the processor uses to indicate additional information about a particular fault subtype. Most of the faults do not use these flags, in which case the flags have no defined values.

The address-of-the-faulting-instruction field contains the IP of the instruction that caused the fault or that was being executed when the fault occurred.

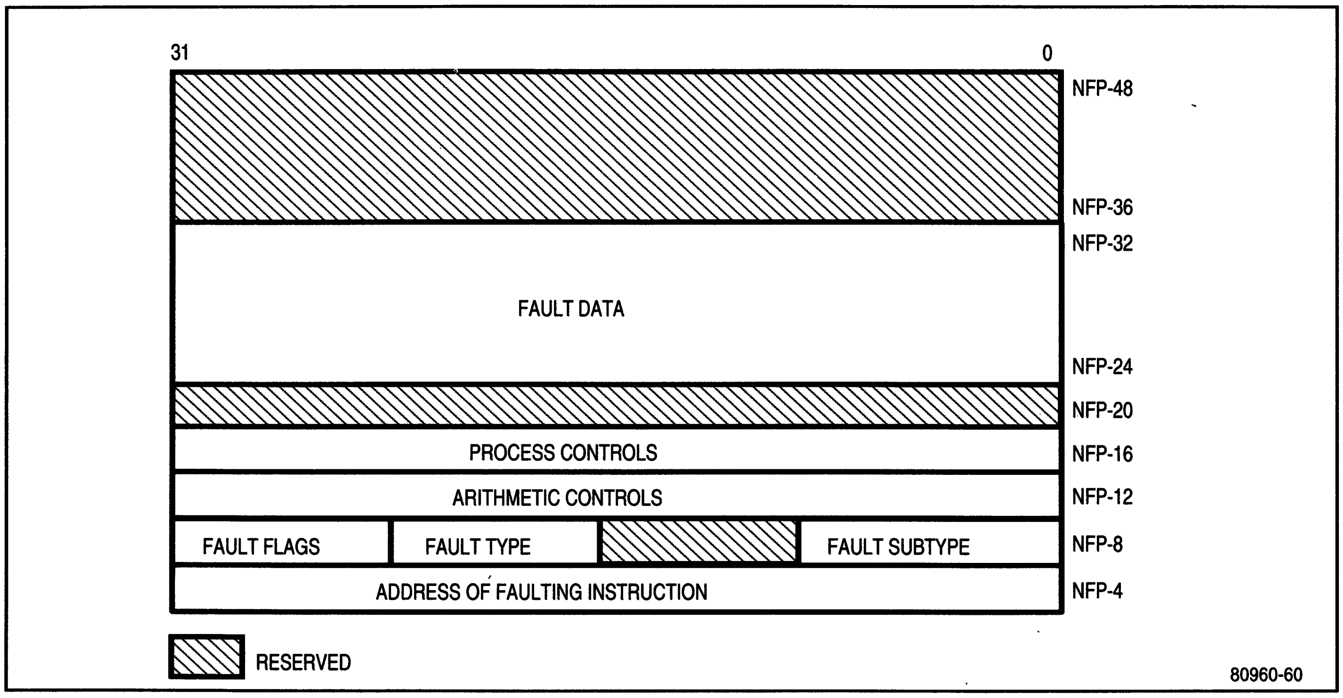

Figure 6-2: Fault Record

The states of the process controls and arithmetic controls at the time that a fault is generated are stored in their respective fields in the fault record. This information is used to resume work on the program after the fault has been handled.

Finally, a three-word fault data field is provided for the fault. The information that is stored in these fields depends on the type of fault that occurs. Any part of a fault-data field that is not used for a particular fault has no defined value. The information that is stored in these fields for each fault type is given in the fault reference section at the end of this chapter.

Saved Instruction Pointer

The saved IP (the RIP that is saved in r2 of the stack frame in use when the fault occurred) is also part of the fault information that the processor saves when a fault occurs. This IP generally points to the next instruction that the processor would have executed if the fault had not occurred, although it may point to the faulting instruction. It is this instruction that the processor begins working on when the return from the fault handler is initiated.

Resumption Record

If the processor suspends an instruction as the result of a fault, it creates a 48-byte resumption record. The criteria that the processor uses to determine whether or not to suspend an instruction and the structure of the resumption record are the same as are used when an interrupt occurs.

Location of the Fault and Resumption Records

The fault and resumption records are stored in the stack that the processor will use to execute the fault-handler procedure. As shown in Figure 6-3, this stack can be the local stack, the supervisor stack, or the interrupt stack. The fault record begins at the byte address of the new frame pointer (NFP) minus 48, and the instruction resumption record begins at NFP minus 96.

FAULT HANDLING ACTION

Once a fault has occurred, the processor saves the program state, calls the fault handler, and restores the program state (if this is possible) once the fault recovery action has been completed. No software other than the fault-handler procedures is required to support this activity.

Four different types of implicit procedure calls can be used to invoke the fault handler, according to the information in the selected fault-table entry: a local call, a local call through the system-procedure table, a supervisor call (also through the system-procedure table), and a supervisor call through the trace table.

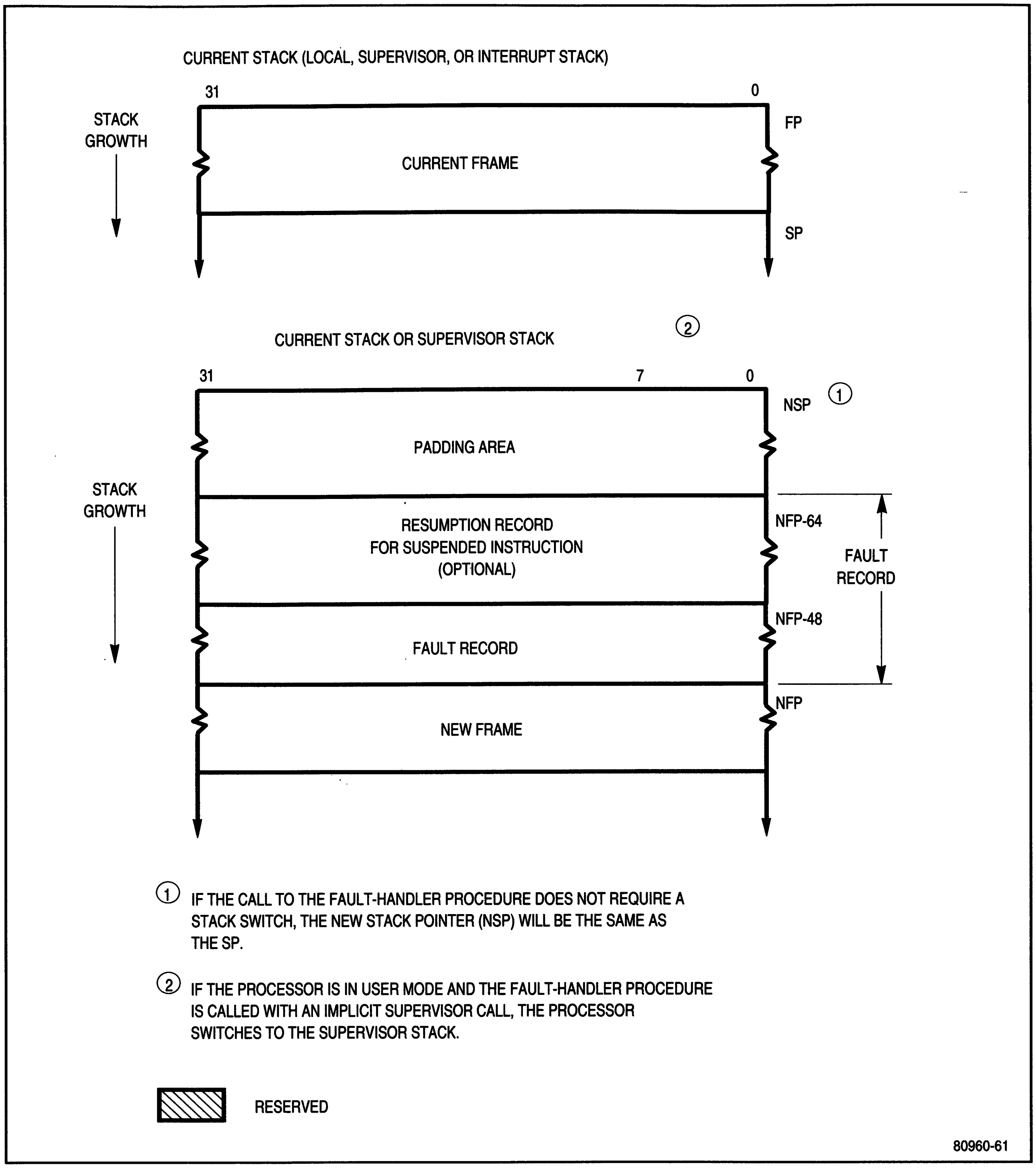
CURRENT STACK (LOCAL, SUPERVISOR, OR INTERRUPT STACK)
31
0
STACK GROWTH
FP
CURRENT FRAME
SP
CURRENT STACK OR SUPERVISOR STACK
②
31
7
0
NSP ①
PADDING AREA
STACK GROWTH
RESUMPTION RECORD FOR SUSPENDED INSTRUCTION (OPTIONAL)
NFP-64
FAULT RECORD
NFP-48
FAULT RECORD
NEW FRAME
NFP
① IF THE CALL TO THE FAULT-HANDLER PROCEDURE DOES NOT REQUIRE A STACK SWITCH, THE NEW STACK POINTER (NSP) WILL BE THE SAME AS THE SP.
② IF THE PROCESSOR IS IN USER MODE AND THE FAULT-HANDLER PROCEDURE IS CALLED WITH AN IMPLICIT SUPERVISOR CALL, THE PROCESSOR SWITCHES TO THE SUPERVISOR STACK.
▨ RESERVED
80960-61

Figure 6-3: Storage of the Fault and Resumption Records on the Stack

Local Call/Return

When the selected fault-handler entry in the fault table is an entry type 00_2 (local procedure), the processor performs the following action:

1. The processor creates a new frame on the stack that the processor is currently using, with the frame-return status field set to 001_2. The stack can be the local stack, the supervisor stack, or the interrupt stack. As shown in Figure 6-3, the new frame pointer (NFP) resides on a 64-byte boundary and provides enough room between the current stack pointer and the NFP for a 48-byte fault record and an optional 16-byte instruction-resumption record. (For local calls to fault handling procedures, the current stack pointer (SP) shown in Figure 6-3 is the same as the new stack pointer (NSP).)
2. The processor stores a fault record as shown in Figure 6-2 on the stack, beginning at NFP minus 48.
3. If the fault caused an instruction to be suspended, the processor includes an instruction-resumption record on the current stack (beginning at NFP minus 96) and sets the resume flag in the saved process controls.
4. Using the procedure address from the selected fault-table entry, the processor performs an implicit call-extended operation to the fault handler.

If the fault handler is not able to perform a recovery action, it performs one of the actions described in the section earlier in this chapter titled "Fault-Handler Procedures."

If the handler action results in a recovery from the fault, a **ret** instruction in the fault handler allows processor control to return to the program that was being worked on when the fault occurred. On the return, the processor performs the following action:

1. The processor copies the arithmetic controls field from the fault record into the arithmetic controls register in the processor.
2. If the resume flag of the process controls is set, the processor reads the resumption record from the stack.
3. The processor deallocates the stack frame created for the fault handler.
4. The processor then resumes work on the program it was working on when the fault occurred, at the instruction in the return IP register.

Local Procedure-Table Call/Return

When the fault-handler entry selects an entry in the system-procedure table (entry type 10_2) and the system-procedure-table entry is for a local procedure, the processor performs the same action as is described in the previous section for a local procedure call/return. The only difference is that the processor gets the address of the fault handler from the system-procedure table rather than from the fault table.

Supervisor Call/Return

When the fault-handler entry selects an entry in the system-procedure table and the system-procedure-table entry is for a supervisor procedure, the processor performs the following actions:

1. If the processor is in user mode when the fault occurs, the processor then reads the supervisor-stack pointer from the system-procedure table and switches to the supervisor stack. The supervisor-stack pointer then becomes the NSP shown in Figure 6-3. Also, the execution mode is set to supervisor.
2. If the processor is already in supervisor mode when the fault occurs, the processor stays on the current stack. Here, the SP and the NSP in Figure 6-3 are the same. Note that if the processor was executing a supervisor procedure when the fault occurred, the current stack will be the supervisor stack; if it was executing an interrupt-handler procedure, the current stack will be the interrupt stack. (The processor switches to supervisor mode when handling interrupts.)
3. The processor creates a new frame on the current stack (as described above for the local call) and stores the fault record and optional instruction resumption record in the areas allocated for them on the stack
4. The processor copies the state of the trace-control flag (byte 12, bit 1) of the system-procedure table into the trace-enable flag field of the process controls, then begins work on the fault-handler procedure selected from the system-procedure table.

On a return from the fault handler, the processor performs the following actions:

1. The processor copies the arithmetic-controls field from the fault record into the arithmetic-controls register in the processor.
2. If the processor is in supervisor mode prior to the return from the fault handler (which it should be), it copies the saved process controls into its internal process controls.
3. If the resume flag of the process controls is set, the processor reads the resumption record from the stack.
4. The processor deallocates the stack frame created for the fault handler and returns to the stack it was using prior to the call to the fault handler routine.
5. If the processor was in user mode prior to the implicit supervisor call, the mode is set back to user mode; otherwise, the processor remains in supervisor mode.
6. The processor resumes work on the program it was working on when the fault occurred, at the instruction in the return IP register.

The restoration of the process controls causes any changes in the process controls through the action of the fault handler to be lost. In particular, if the **ret** instruction from the fault handler caused the trace-fault-pending flag in the process controls to be set, this setting would be lost on the return.

Trace-Fault-Handler Procedure-Table Call/Return

When the fault table entry is for a trace fault, the processor performs the same action as is described in the previous section for a supervisor call and return. The only difference is that the processor uses the trace table instead of the system-procedure table.

PRECISE AND IMPRECISE FAULTS

NOTE

This section describes a feature of the i960 architecture that is not implemented in the 80960SA/SB processor. It is described here for readers interested in making applications portable to other implementations of the architecture. In the 80960SA/SB processor, all faults are precise (i.e., imprecise faults are not allowed to be generated). Also, the NIF flag in the arithmetic controls has no effect (the processor assumes it is set to 1), and the **syncf** instruction acts as a no-op.

The i960 architecture supports concurrent and out-of-order execution of instructions. When an implementation provides this feature and two or more instructions are being executed concurrently, it is possible for the instructions to generate faults simultaneously. When this situation occurs, it may not be possible to recover from some of the faults, because the state of the instructions surrounding the faulting instructions has changed or the saved IP is unpredictable.

The processor provides two mechanisms to allow the circumstances under which faults are generated to be controlled. These mechanisms are the no-imprecise-faults (NIF) flag in the arithmetic controls and the synchronize faults instruction (**syncf**). The following paragraphs describe how these mechanisms can be used.

Faults are grouped into the following categories: precise, imprecise, and asynchronous. Precise faults are those that are intended to be recoverable by software. For any instruction that can generate a precise fault, the processor will (1) not execute the instruction if an unfinished prior instruction will fault and (2) not execute subsequent out-of-order instructions that will fault. The following faults are always precise:

- trace
- protection

Imprecise faults are those where the architecture does not guarantee that sufficient information is saved in the fault record to allow recovery from the fault. For imprecise faults, the address of the faulting instruction is correct, but the state of execution of the instructions surrounding the faulting instruction may be unpredictable. Also, the architecture allows imprecise faults to be generated out of order, which means that the saved IP may not be of any value for recovery. The faults that in some circumstances are allowed to be imprecise include the following:

- operation
- arithmetic
- floating point
- constraint
- type

Asynchronous faults are those whose occurrence has no direct relationship to the instruction pointer. This category includes the machine fault.

The NIF flag controls whether or not imprecise faults are allowed. When this flag is set, all faults must be precise. This means that the following conditions hold true:

1. All faults are generated in order.
2. A precise fault record is provided for each fault (i.e., the address of the faulting instruction is correct and the saved IP provides a valid reentry point into the program).

In the no-imprecise-faults mode, the ability to execute instructions concurrently is essentially disabled.

When the NIF flag is clear, faults in the imprecise category are allowed to be generated imprecisely (i.e., in parallel and out of order, and with an imprecise fault record). Here, the following conditions hold true:

1. When an imprecise fault is generated, the address of the faulting instruction in the fault record is valid, but the saved IP is unpredictable.
2. If instructions are executed out-of-order and multiple imprecise faults occur, recovery from some of the faults may not be possible, because the source operands of the faulting instructions may be modified when subsequent instructions are executed out of order.

The **syncf** instruction forces the processor to complete execution of all instructions that occur prior to the **syncf** instruction and to generate all faults, before it begins work on instructions that occur after the **syncf** instruction. This instruction has two uses. One use is to force faults to be precise when the NIF is clear. The other use is to insure that all instructions are complete and all faults generated in one block of code before execution of another block of code (for example, on Ada block boundaries when the blocks have different exception handlers).

The intent of these fault-generating modes is that compiled code should execute with the NIF clear, using the **syncf** instruction where necessary to ensure that faults occur in order. In this mode, imprecise faults are considered as catastrophic errors from which recovery is not needed.

If recovery from one or more of the imprecise faults is required (for example, a program that needs to handle unmasked floating-point exceptions and recover from them) and the fault handler cannot be closely coupled with the application to perform recovery even if the faults are imprecise, the NIF should be set. Executing with the NIF set will likely lead to slower execution times.

FAULT REFERENCE

This section describes each of the fault types and subtypes and gives detailed information about what is stored in the various fields of the fault record. The section is organized alphabetically by fault type.

Fault Reference Notation

The following paragraphs describe the information that is provided for each fault type.

Fault Type and Subtype

The fault-type section gives the number entered in the fault-type field of the fault record for the given fault type. The fault-subtype section lists the fault subtypes and their associated number or bit position in the fault-subtype field of the fault record.

Function

The function section gives a general description of the purpose of the fault type, then describes the purpose of each of the fault subtypes in detail. It also describes how the processor handles each fault subtype.

Fault Record

The fault record section describes how the flags, fault-data, and address-of-faulting-instruction fields of the fault record are used for the fault type and subtypes.

Saved IP

The saved IP section describes what value is saved in the RIP register (r2) of the stack frame the processor was using when the fault occurred.

Program State Changes

The program state changes section describes the effects that the fault subtypes have on the control flow of a program.

Arithmetic Faults

Fault Type: 3_{16}

Fault Subtype:

$\textbf{Number}_{16}$	**Name**
0	Reserved
1	Integer Overflow
2	Arithmetic Zero-Divide
3-F	Reserved

Function: Indicates that there is a problem with an operand or the result of an arithmetic instruction. This fault type applies only to ordinal and integer instruction, not floating-point instructions.

The integer-overflow fault occurs when the result of an integer instruction overflows the destination and the integer-overflow mask in the arithmetic-controls register is cleared. Here, the *n* least significant bits of the result are stored in the destination, where *n* is the destination size.

The arithmetic zero-divide fault occurs when the divisor operand of an ordinal or integer divide instruction is zero.

Fault Record:

Flags:	Not used.
Fault Data:	Not used.
Addr. Fault. Inst.:	IP for the instruction on which the processor faulted.

Saved IP: IP for the instruction that would have been executed next, if the fault had not occurred.

Prog. State Changes: A change in the program's control flow accompanies the integer-overflow fault, because the result is stored in the destination before the fault is generated. The faulting instruction can thus not be reexecuted.

A change in the program's control flow does not accompany the arithmetic zero-divide fault, because the fault occurs before the execution of the faulting instruction.

Constraint Faults

Fault Type: 5_{16}

Fault Subtype:

Number$_{16}$	**Name**
0	Reserved
1	Constraint Range
2	Privileged
3-F	Reserved

Function: Indicates that the processor is either in or not in the required state for the instruction to be executed.

The constraint-range fault occurs when a fault-if instruction is executed and the condition code in the arithmetic controls matches the condition required by the instruction.

The privileged fault is generated when a program or procedure attempts to use a privileged (supervisor-mode only) instruction, while the processor is in user mode.

Fault Record:

Flags:	Not used.
Fault Data:	Not used.
Addr. Fault. Inst.:	IP for the instruction on which the processor faulted

Saved IP: Not used.

Prog. State Changes: No changes in the program's control flow accompany the constraint-range fault. This fault occurs after the fault-if instruction has been executed, but the instruction has no effect on the program state.

Floating-Point Faults

Fault Type: 4_{16}

Fault Subtype:

Bit Number	**Name**
Bit 0	Floating Overflow
Bit 1	Floating Underflow
Bit 2	Floating Invalid-Operation
Bit 3	Floating Zero-Divide
Bit 4	Floating Inexact
Bit 5	Floating Reserved-Encoding
Bits 6 and 7	Reserved

Function: Indicates that there is a problem with an operand or the result of a floating-point instruction. Each floating-point fault is assigned a bit in the fault-subtype field. Multiple floating-point faults can only occur simultaneously, however, with the floating-overflow, floating-underflow, and floating-inexact faults.

The floating-point faults are described in detail in the section in Chapter 10 titled "Exceptions and Fault Handling." The following paragraphs give a brief description of each floating-point fault.

A floating-overflow fault occurs when (1) the floating-point overflow mask is clear and (2) the infinitely precise result of a floating-point instruction exceeds the largest allowable finite value for the specified destination format. This fault interacts with the floating-inexact fault (as described in Chapter 10).

A floating-underflow fault occurs when (1) the floating-point underflow mask is clear and (2) the infinitely precise result of a floating-point instruction is less than the smallest possible normalized, finite value for the specified destination format. This fault interacts with the floating-inexact fault (as described in Chapter 10).

The floating invalid-operation fault occurs when (1) the floating-point invalid-operation mask is clear and (2) one of the source operands for a floating-point instruction is inappropriate for the type of operation being performed.

The floating zero-divide fault occurs when (1) the floating-point zero-divide mask is clear and (2) the divisor operand of a floating-point divide instruction is zero.

The floating-inexact fault occurs when (1) the floating-point inexact mask is clear and (2) an infinitely precise result cannot be encoded in the format specified for the destination operand. This fault interacts with the floating-overflow and floating-underflow faults (as described in Chapter 10).

The floating reserved-encoding fault occurs when a denormalized value is used as an operand in a floating-point instruction and the normalizing-mode bit in the arithmetic controls is clear.

Fault Record:	**Flags:**	**F0** -- Used if inexact fault occurs in conjunction with overflow or underflow fault. If set, F0 indicates that the adjusted result has been rounded toward +∞; if clear, F0 indicates that the adjusted result has been rounded toward -∞.
		F1 -- Used with overflow and underflow faults only. If set, F1 indicates that the adjusted result has been bias adjusted, because its exponent was outside the range of the extended-real format.
	Fault Data:	Used only with overflow and underflow faults. Adjusted result is stored in this field in extended-real format (as shown in Figure 10-5).
	Addr. Fault. Inst.:	IP for the instruction on which the processor faulted.

Saved IP: IP for the instruction that would have been executed next, if the fault had not occurred.

Prog. State Changes: Changes in the program's control flow accompany the floating-overflow, floating-underflow, and floating-inexact faults, because a result is stored in the destination before the fault is generated. The faulting instruction can thus not be reexecuted.

Changes in the program's control flow do not accompany the floating invalid-operation, floating zero-divide, and floating reserved-encoding faults, because the faults occur before the execution of the faulting instruction.

Operation Faults

Fault Type: 2_{16}

Fault Subtype:

Number$_{16}$	**Name**
0	Reserved
1	Invalid Opcode
2	Reserved
3	Reserved
4	Invalid Operand
5 - F	Reserved

Function: Indicates that the processor cannot execute the current instruction because of invalid instruction syntax or operand semantics.

The invalid-opcode fault occurs when the processor attempts to execute an instruction that contains an undefined opcode or addressing mode.

The invalid-operand fault occurs when the processor attempts to execute an instruction for which one or more of the operands have special requirements and one or more of the operands do not meet these requirements. This fault subtype is not generated on floating-point instructions.

Fault Record:

Flags:	Not used.
Fault Data:	Not used.
Addr. Fault. Inst.:	IP for the instruction on which the processor faulted.

Saved IP: Not used.

Prog. State Changes: A change in the program's control flow does not accompany the operation faults, because the faults occur before the execution of the faulting instruction.

Protection Faults

Fault Type:	7_{16}	
Fault Subtype:	**Bit Number**	**Name**
	Bit 0	Reserved
	Bit 1	Length
	Bit 2-7	Reserved

Function: Indicates that the index operand used in a **calls** instruction points to an entry beyond the extent of the system-procedure table.

Fault Flags:	Not used.
Fault Data:	Not used.
Addr. Fault. Inst.:	IP for the instruction on which the processor faulted.

Saved IP: Same as the address-of-faulting-instruction field.

Prog. State Changes: A change in the program's control flow does not accompany the protection length fault.

Trace Faults

Fault Type: 1_{16}

Fault Subtype:

Bit Number	**Name**
Bit 0	Reserved
Bit 1	Instruction Trace
Bit 2	Branch Trace
Bit 3	Call Trace
Bit 4	Return Trace
Bit 5	Prereturn Trace
Bit 6	Supervisor Trace
Bit 7	Breakpoint Trace

Function:

Indicates that the processor has detected one or more trace events. The processor's event tracing mechanism is described in detail in Chapter 7.

A trace event is the occurrence of a particular instruction or type of instruction in the instruction stream. The processor recognizes seven different trace events (instruction, branch, call, return, prereturn, supervisor, and breakpoint). It detects these events, however, only if a mode bit is set for the event in the trace controls word, which is cached in the processor chip. If, in addition, the trace-enable flag in the process controls is set, the processor generates a fault when a trace event is detected.

The fault is generated following the instruction that causes a trace event (or prior to the instruction for the prereturn trace event).

The following trace modes are available:

- **Instruction** -- Generate trace event following any instruction.
- **Branch** -- Generate trace event following any branch instruction when branch is taken. (Does not occur on branch and link and call instructions.)
- **Call** -- Generate trace event following any call or branch-and-link instruction, or implicit procedure call (i.e., call to fault or interrupt handler).
- **Return** -- Generate trace event following any return instruction.
- **Prereturn** -- Generate trace event prior to any return instruction, providing the prereturn-trace flag in r0 is set. (The processor sets this flag automatically when prereturn tracing is enabled.)

- **Supervisor** -- Generate trace event following any call-system instruction that references a supervisor procedure entry in the system-procedure table.
- **Breakpoint** -- Generate trace event following any processor action that causes a breakpoint condition (such as a **mark** or **fmark** instruction).

There is a trace fault subtype and a bit in the fault-subtype field associated with each of these modes. Multiple fault subtypes can occur simultaneously, with the fault-subtype bit set for each subtype that occurs.

When a fault type other than a trace fault occurs during the execution of an instruction that causes a trace event, the non-trace-fault is handled before the trace fault. An exception to this rule is the prereturn trace fault. The prereturn trace fault will occur before the processor has a chance to detect a non-trace-fault, so it is handled first.

Likewise, if an interrupt occurs during an instruction that causes a trace event, the interrupt is serviced before the trace fault is handled. Again, the prereturn trace fault is an exception. Since it occurs before the instruction, it will be handled before any interrupt that might occur during the execution of the instruction.

Fault Record:

Flags:	Not used.
Fault Data:	Not used.
Addr. Fault. Inst.:	IP for the instruction that caused the trace event, except for the prereturn trace fault. For the prereturn trace fault, this field has no defined value.

Saved IP: IP for the instruction that would have been executed next, if the fault had not occurred.

Prog. State Changes: A change in the program's control flow accompanies all the trace faults (except the prereturn trace fault), because the events that can cause a trace fault occur after the faulting instruction is completed. As a result, the faulting instruction cannot be reexecuted upon returning from the fault handler.

Since the prereturn trace fault occurs before the **ret** instruction is executed, a change in the program's control flow does not accompany this fault and the faulting instruction can be executed upon returning from the fault handler.

Type Faults

Fault Type:	A_{16}	
Fault Subtype:	**Number$_{16}$**	**Name**
	0	Reserved
	1	Type Mismatch
	2-F	Reserved
Function:	Indicates that an attempt was made to execute the **modpc** instruction while the processor was in the user mode.	
Fault Record:	**Flags:**	Not used.
	Fault Data:	Not used.
	Addr. Fault. Inst.:	IP for the instruction on which the processor faulted.
Saved IP:	Not used.	
Prog. State Changes:	When a type mismatch fault occurs, the accompanying state of the program is undefined. The processor is thus not able to return predictably from the fault handler to the point in the program where the fault occurred.	