Debugging 7

CHAPTER 7
DEBUGGING

This chapter describes the tracing facilities of the 80960SA/SB processor, which allow the monitoring of instruction execution.

OVERVIEW OF THE TRACE-CONTROL FACILITIES

The 80960SA/SB processor provides facilities for monitoring the activity of the processor by means of trace events. A trace event in the 80960SA/SB is a condition where the processor has just completed executing a particular instruction or type of instruction, or where the processor is about to execute a particular instruction.

By monitoring trace events, debugging software is able to display or analyze the activity of the processor or of a program. This analysis can be used to locate software or hardware bugs or for general system monitoring during the development of system or applications programs.

The typical way to use this tracing capability is to set the processor to detect certain trace events either by means of the trace-controls word or a set of breakpoint registers. An alternate method of creating a trace event is with the **mark** and force mark **(fmark)** instructions. These instructions cause an explicit trace event to be generated when the processor detects them in the instruction stream.

If tracing is enabled, the processor signals a trace fault when it detects a trace event. The fault handler for trace faults can then call the debugging monitor software to display or analyze the state of the processor when the trace event occurred.

REQUIRED SOFTWARE SUPPORT FOR TRACING

To use the processor's tracing facilities, software must provide trace-fault handling procedures, perhaps interfaced with a debugging monitor. Software must also manipulate several control flags to enable the various tracing modes and to enable or disable tracing in general. These control flags are located in the system-data structures described in the next section.

TRACE CONTROLS

The following flags or fields control tracing:

- Trace controls
- Trace-enable flag in the process controls
- Trace-fault-pending flag in the process controls
- Trace flag (bit 0) in the return-status field of register r0
- Trace-control flag in the supervisor-stack-pointer field of the system table or a procedure table .

Trace-Controls Word

The trace-controls word is cached internally in the processor.

The trace controls allow software to define the conditions under which trace events are generated. Figure 7-1 shows the structure of the trace-controls word.

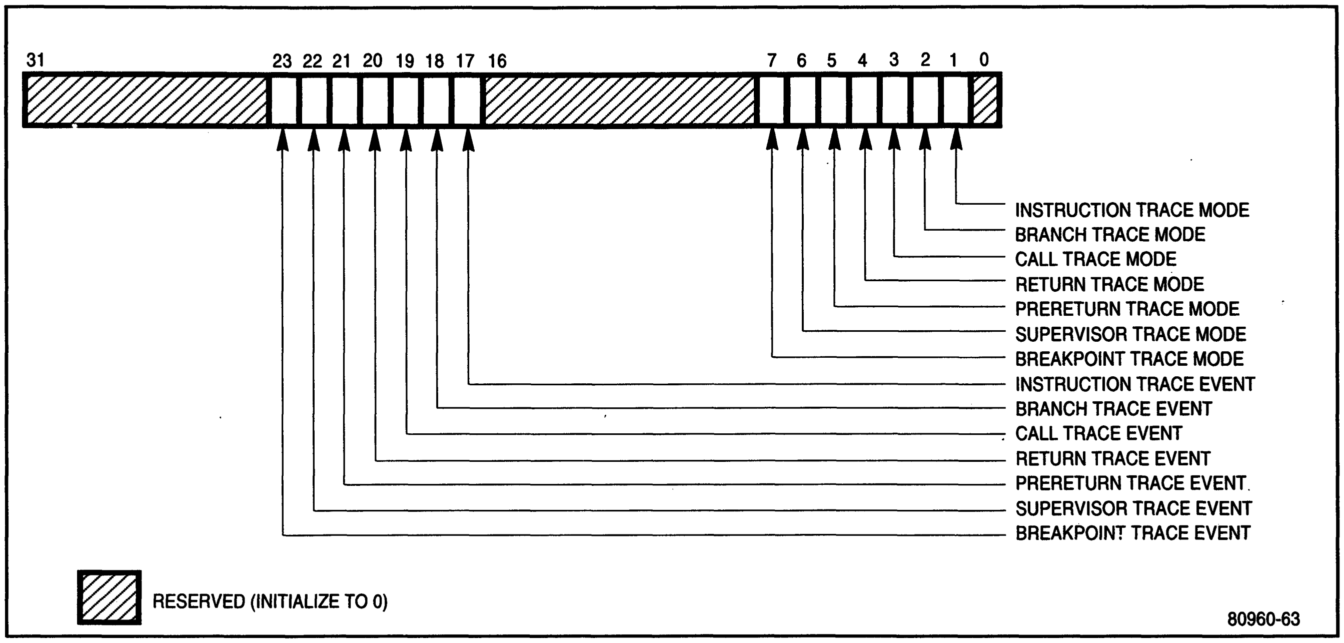

Figure 7-1: Trace-Controls Word

This word contains two sets of bits: the mode flags and the event flags. The mode flags define a set of trace modes that the processor can use to generate trace events. A mode represents a subset of instructions that will cause trace events to be generated. For example, when the call-trace mode is enabled, the processor generates a trace event whenever a call or branch-and-link operation is executed. To enable a trace mode, the kernel sets the mode flag for the selected trace mode in the trace controls. The trace modes are described later in this chapter.

The processor uses the event flags to keep track of which trace events (for those trace modes that have been enabled) have been detected.

A special instruction, the modify-trace-controls (**modtc**) instruction, allows software to set or clear flags in the trace controls. On initialization, all the flags in the processor's internal trace controls are cleared. The **modtc** instruction can then be used to set or clear trace mode flags as required.

Software can access the event flags using the **modtc** instruction; however, there is no reason to. The processor modifies these flags as part of its trace-handling mechanism.

Bits 0, 8 through 16, and 24 through 31 of the trace controls are reserved. Software should initialize these bits to zero and not modify them. Note that the processor may modify these fields for internal state storage.

Trace-Enable and Trace-Fault-Pending Flags

The trace-enable flag and the trace-fault-pending flag, located in the process controls (shown in Figure 3-2), control tracing. The trace-enable flag enables the processor's tracing facilities. When this flag is set, the processor generates trace faults on all trace events.

Typically, software selects the trace modes to be used through the trace controls. It then sets the trace-enable flag when tracing is to begin. This flag is also altered as part of some of the call and return operations that the processor carries out, as described at the end of this chapter.

The trace-fault-pending flag allows the processor to keep track of the fact that an enabled trace event has been detected. The processor uses this flag as follows. When the processor detects an enabled trace event, it sets this flag. Before executing an instruction, the processor checks this flag. If the flag is set, it signals a trace fault. By providing a means of recording the occurrence of a trace event, the trace-fault-pending flag allows the processor to service an interrupt or handle a fault other than a trace fault, before handling the trace fault. Software should not modify this flag.

Trace Control on Supervisor Calls

The trace flag and the trace-control flag allow tracing to be enabled or disabled when a call-system instruction (**calls**) is executed that results in a switch to supervisor mode. This action occurs independent of whether or not tracing is enabled prior to the call.

When a supervisor call is executed (**calls** instruction that references an entry in the system-procedure table with an entry type 11_2), the processor saves the current state of the trace-enable flag (from the process controls) in the trace flag (bit 0) of the return-status field of register r0.

Then, when the processor selects the supervisor procedure from the procedure table, it sets the trace-enable flag in the process controls according to the setting in the trace-control flag in the procedure table (bit 0 of the word that contains the supervisor-stack pointer). When the trace-control flag is set, tracing is enabled; when it is clear, tracing is disabled.

On a return from the supervisor procedure, the trace-enable flag in the process controls is restored to the value saved in the return-status field of register r0.

TRACE MODES

The following trace modes can be enabled through the trace controls:

- Instruction trace
- Branch trace
- Call trace
- Return trace
- Prereturn trace
- Supervisor trace
- Breakpoint trace

These modes can be enabled individually or several modes can be enabled at once. Some of these modes overlap, such as the call-trace mode and the supervisor-trace mode. The section later in this chapter titled "Handling Multiple Trace Events" describes what the processor does when multiple trace events occur.

The following sections describe each of the trace modes.

Instruction Trace

When the instruction-trace mode is enabled, the processor generates an instruction-trace event each time an instruction is executed. The event is generated immediately after the instruction is executed. This mode can be used within a debugging monitor to single-step the processor.

Branch Trace

When the branch-trace mode is enabled, the processor generates a branch-trace event immediately after a branch instruction that branches is executed. A branch-trace event is not generated for conditional-branch instructions that do not branch. Also, branch-and-link, call, and return instructions do not cause branch-trace events to be generated.

Call Trace

When the call-trace mode is enabled, the processor generates a call-trace event immediately after a call instruction (**call**, **callx**, or **calls**) or a branch-and-link instruction (**bal** or **balx**) is executed. An implicit call, such as the action used to invoke a fault handler or an interrupt handler, also causes a call-trace event to be generated.

When the processor detects a call-trace event, it also sets the prereturn-trace flag (bit 3 of register r0) in the new frame created by the call operation or in the current frame if a branch-and-link operation was performed. The processor uses this flag to determine whether or not to signal a prereturn-trace event on a **ret** instruction.

Return Trace

When the return-trace mode is enabled, the processor generates a return-trace event any time a **ret** instruction is executed.

Prereturn Trace

The prereturn-trace mode causes the processor to generate a prereturn-trace event prior to the execution of any **ret** instruction, providing the prereturn-trace flag in r0 is set. (Prereturn tracing cannot be used without enabling call tracing.)

The processor sets the prereturn-trace flag whenever it detects a call-trace event (as described above for the call-trace mode). This flag performs a prereturn-trace-pending function. If another trace event occurs at the same time as the prereturn-trace event, the prereturn-trace flag allows the processor to fault on the non-prereturn-trace event first, then come back and fault again on the prereturn-trace event. The prereturn trace is the only trace event that can cause two successive trace faults to be generated between instruction boundaries.

Supervisor Trace

When the supervisor-trace mode is enabled, the processor generates a supervisor-trace event any time (1) a call-system instruction (**calls**) is executed, where the procedure table entry is a supervisor procedure, or (2) when a **ret** instruction is executed and the return-status field is set to 010_2 or 011_2 (i.e., return from supervisor mode).

This trace mode allows a debugging program to determine the boundaries of kernel-procedure calls within the instruction stream.

Breakpoint Trace

The breakpoint-trace mode allows trace events to be generated at places other than those specified with the other trace modes. This mode is used in conjunction with the **mark** and force-mark (**fmark**) instructions, and the breakpoint registers.

The **mark** and **fmark** instructions allow breakpoint-trace events to be generated at specific points in the instruction stream. When the breakpoint-trace mode is enabled, the processor generates a breakpoint-trace event any time it encounters a **mark** instruction. The **fmark** causes the processor to generate a breakpoint-trace event regardless of whether the breakpoint-trace mode is enabled or not. Both of these instructions require that the trace-enable flag in the process-controls word be set for them to function correctly. If the trace-enable flag is clear, the instructions behave as no-ops.

The processor has two, one-word breakpoint registers, designated as breakpoint 0 and breakpoint 1. Using the set-breakpoint-register IAC, one instruction pointer can be loaded into each register. The processor then generates a breakpoint trace any time it executes an instruction referenced in a breakpoint register.

TRACE-FAULT HANDLER

A fault handler is a procedure that the processor calls to handle faults that occur. The requirements for fault handlers are given in Chapter 6 in the section titled "Fault-Handler Procedures."

A trace-fault handler has one additional restriction. It must be called through the trace table with an implicit supervisor call, and the trace-control flag in the trace-table entry must be clear. This restriction insures that tracing is turned off when a trace fault is being handled, which is necessary to prevent an endless trace-fault loop.

SIGNALING A TRACE EVENT

To summarize the information presented in the previous sections, the processor signals a trace event when it detects any of the following conditions:

- An instruction included in a trace-mode group is executed (all trace events except prereturn) or about to be executed (in the case of a prereturn-trace event) and the trace mode for that instruction is enabled.
- An implicit call operation has been executed and the call-trace mode is enabled.

- A **mark** instruction has been executed and the breakpoint-trace mode is enabled.
- An **fmark** instruction has been executed.
- An instruction specified in a breakpoint register is executed and the breakpoint-trace mode is enabled.

When the processor detects a trace event and the trace-enable flag in the process controls is set, the processor performs the following action:

1. The processor sets the appropriate trace-event flag in the trace controls. If a trace event meets the conditions of more than one of the enabled trace modes, a trace-event flag is set for each trace mode condition that is met.
2. The processor sets the trace-fault-pending flag in the process controls.

If, when the processor detects a trace event, the trace-enable flag in the process controls is clear, the processor sets the appropriate event flags, but does not set the trace-fault-pending flag.

HANDLING MULTIPLE TRACE EVENTS

If the processor detects multiple trace events, it records one or more of them based on the following precedence, where 1 is the highest precedence:

1. Supervisor-trace event
2. Breakpoint- (from **mark** or **fmark** instruction, or from a breakpoint register), branch-, call-, or return-trace event
3. Instruction-trace event

When multiple trace events are detected, the processor may not signal each event; however, it will signal at least the one with the highest precedence.

TRACE-HANDLING ACTION

Once a trace event has been signaled, the processor determines how to handle the trace event, according to the setting of the trace-enable and trace-fault-pending flags in the process controls and to other events that might occur simultaneously with the trace event such as an interrupt or a non-trace fault.

The following sections describe how the processor handles trace events for various situations.

Normal Handling of Trace Events

Prior to executing an instruction, the processor performs the following action regarding trace events:

1. The processor checks the state of the trace-fault pending flag. If this flag is clear, the processor begins execution of the next instruction. If the flag is set, the processor performs the following actions.
2. The processor checks the state of the trace-enable flag. If the trace-enable flag is clear, the processor clears any trace event flags that have been set, prior to starting execution of the next instruction. If the trace-enable flag is set, the processor performs the following action.
3. The processor signals a trace fault and begins the fault handling action, as described in Chapter 6.

Prereturn-Trace Handling

The processor handles a prereturn-trace event the same as described above except when it occurs at the same time as a non-trace fault. Here, the non-trace fault is handled first.

On returning from the fault handler for the non-trace fault, the processor checks the prereturn-trace flag in register r0. If this flag is set, the processor generates a prereturn-trace event, then handles it as described above.

Tracing and Interrupt Handlers

When the processor invokes an interrupt handler to service an interrupt, it disables tracing. It does this by saving the current state of the process controls, then clearing the trace-enable and trace-fault-pending flags in the current process controls.

On returning from the interrupt handler, the processor restores the process controls to the state they were in prior to handling the interrupt, which restores the state of the trace-enable and trace-fault-pending flags. If these two flags were set prior to calling the interrupt handler, a trace fault will be signaled on the return from the interrupt handler.

Tracing and Fault Handlers

The processor can invoke a fault handler with either an implicit local call or an implicit supervisor call. On a local call, the trace-enable and trace-fault-pending flags are neither saved on the call nor restored on the return. The state of these flags on the return is thus dependent on the action of the fault handler.

On a supervisor call, the trace-enable and trace-fault-pending flags are saved, as part of the saved process controls, and restored on the return. So, if these two flags were set prior to calling the fault handler, a trace fault will be signaled on the return from the fault handler.

NOTE

On a return from an interrupt handler, the trace-fault-pending flag is restored. If this flag is set as a result of the handler's **ret** instruction (i.e., indicating a return trace event), the detected trace event is lost.

The action described above is also true on a return from a fault handler, when the fault handler has been called with an implicit supervisor call.