

DEBUG CIRCUIT COMPARING PROCESSOR INSTRUCTION SET OPERATING MODE

FIELD

[0001] The present invention relates generally to the field of processors and in particular to a system and method of debugging code on a processor via a debugging circuit that includes consideration of the processor instruction set operating mode.

BACKGROUND

[0002] Modern processors are extremely complicated systems. Most modern processors employ a pipelined architecture, where sequential instructions, each having multiple execution steps, are overlapped in execution. Many processors, known as "superscalar" processors, include two or more separate pipelines for parallel instruction execution. To avoid stalls in the pipeline due to branch instructions, most processors employ various forms of branch prediction, with speculative fetching and execution of instructions when branches are predicted taken. To improve performance, many modern processors separate logical register addresses from the corresponding physical memory storage registers, known as register renaming. To provide programs the illusion of unlimited, fast memory, many processors execute code in a virtual address space, translating addresses to one or more physical address spaces as data traverses a memory hierarchy (e.g., register, cache, main memory). With such complexity, the execution of code within a modern processor is very difficult to accurately track and validate. In particular, anomalies may be very hard to debug.

[0003] In addition, the software that executes on modern processors is itself extremely complex. With the advent of Reduced Instruction Set Computing (RISC) processor architectures, much of the computational and logical complexity of computing migrated from processor instructions to optimizing compilers. That is,

compilers build complex operations from a relatively small set of processor instructions, each of which is optimized for a particular, specific function. This results in a more lengthy and complex sequence of instructions, including, e.g., logical, arithmetic, load/store, and branch operations, for a given computational task. Such complex code can be difficult to debug when errors cause anomalous program behavior.

[0004] To assist in the daunting task of debugging complex code executing on a complex processor, debugging tools are built into many processors. These may comprise address and data comparators for identifying specific instructions and/or data patterns. The debug tools may additionally include address range comparators, so that the debugging or tracing may be limited to predetermined code segments. Other debug tools may include counters, sequencers, and the like, to provide flexibility in specifying the conditions that specify breakpoints and/or trace triggers. Trace information may be provided off-chip, such as via a dedicated bus, or may be stored in a dedicated on-chip buffer. Breakpoints may trigger one or more external signals, such as to arm and/or trigger a logic analyzer or illuminate an LED; may cause an exception, branching code execution to a debugging routine; or may simply halt execution, allowing the contents of various registers and caches to be inspected. These debugging tools, which are not utilized during normal processor execution, are designed into the processor circuits in such a way as to minimize impact on processor performance and power consumption.

[0005] Programmers may explicitly set the proper instruction set operating mode in software, prior to executing instructions according to the relevant instruction set encoding. However, errant software may at times branch to a particular instruction address that was intended to be executed while in one instruction set operating mode, but while the processor is actually in some other instruction set operating mode. In such a case, the processor may attempt to execute the instruction at that address using the improper instruction set decoding, leading to incorrect results.

[0006] For example, some versions of the ARM processor architecture include at least two instruction set operating modes: a 32-bit ARM mode and a 16-bit Thumb mode. Table 1 lists a code snippet of ARM instructions:

000096F0	cpy	r4, r0
000096F4	bl	0xA8A0
000096FC	cpy	r1, r4
00009700	ldr	r0, [r0, #0x8]

Table 1: ARM mode instruction encoding

[0007] Table 2 lists the same code interpreted in Thumb mode:

000096F0	and	r0, r0
000096F2	b	0x9A36
000096F4	lsl	r1, r5, #0x11
000096F6	add.w	r0, r0, r4, lsl #0x4
000096FA	b	0x9A3E
000096FC	lsl	r0, r1, #0x0
000096FE	b	0x9222
00009700	and	r0, r2

Table 2: Thumb mode instruction encoding

[0008] Note, in particular, the branch instructions at 96F2, 96FA, and 96FE. Due to erratic branching, it may be difficult to ascertain the point at which an error caused the ARM code of Table 1 to be interpreted in Thumb mode as in Table 2, and hence to debug the error. This difficulty arises because prior art debug circuits do not include the processor instruction set operating mode as an input to the logic that triggers breakpoints, initiates traces, and of the like.

[0009] Most processors that support two or more operating “modes,” such as supervisor and user modes, or real and protected modes, switch between the modes by causing an exception and branching to a mode switching routine. This operation is easily detected using prior art debugging tools and software by setting a breakpoint or beginning a trace on the instruction address of the mode switching routine. A processor switching instruction set operating modes without causing an exception is one example of code behavior that is difficult to diagnose with prior art debugging tools,

that do not include the processor instruction set operating mode as a consideration in triggering breakpoints, initiating traces, and the like.

SUMMARY

[0010] According to one or more embodiments, a debug circuit in a processor operative to execute two or more instruction sets identifies the address at which the processor switches from one instruction set operating mode to a different instruction set operating mode, and outputs an alert or indication of the mode switch. The alert or indication may comprise a breakpoint signal that halts execution and/or is output as an external signal of the processor. The instruction address at which the processor switches instruction set operating modes may additionally be output. Alternatively, the alert or indication may comprise starting or stopping a trace operation, causing an exception, or any other known debugger function.

[0011] One embodiment relates to a method of debugging software code comprising a plurality of instructions on a processor having at least two different instruction set operating modes. A target instruction set operating mode is received prior to executing any code instructions. Code instructions are executed and, for each instruction executed, the current processor instruction set operating mode is compared to the target instruction set operating mode. An alert is triggered if the current processor instruction set operating mode matches the target instruction set operating mode.

[0012] Another embodiment relates to a method of executing software on a processor having at least first and second instruction set operating modes. The address at which the processor switches from the first instruction set operating mode to the second instruction set operating mode is identified, and an alert is triggered in response to the identification.

[0013] Yet another embodiment relates to a processor operative to execute instructions according to two or more instruction set encodings, each in a different instruction set operating mode. The processor includes a current instruction set operating mode indicator and a data storage location operative to store a target instruction set operating mode indicator. The processor also includes an execution unit operative to execute instructions according to a current instruction set operating mode and a compare circuit operative to compare the current instruction set operating mode to the target instruction set operating mode upon executing each instruction, and to output an indication if the current instruction set operating mode matches the target instruction set operating mode.

[0014] Still another embodiment relates to a processor operative to execute two or more instruction set encodings, each in a different instruction set operating mode. The processor includes a compare circuit operative to detect a change in the instruction set operating mode and to output an indication of the instruction set operating mode change, and the instruction address at which the change occurs, in response to the detection.

BRIEF DESCRIPTION OF DRAWINGS

[0015] Figure 1 is a functional block diagram of a processor.

[0016] Figure 2 is a functional block diagram of a processor pipeline, including a debug circuit.

[0017] Figure 3 is a flow diagram of a diagnostic/debug process.

DETAILED DESCRIPTION

[0018] Figure 1 depicts a functional block diagram of a processor 10. The processor 10 executes instructions in an instruction execution pipeline 12 according to control logic 14. The control logic 14 maintains a Program Counter (PC) 15, and sets and clears bits in one or more status registers 16 to indicate, e.g., the current

instruction set operating mode, information regarding the results of arithmetic operations and logical comparisons (zero, carry, equal, not equal), and the like. In some embodiments, the pipeline 12 may be a superscalar design, with multiple, parallel pipelines. The pipeline 12 may also be referred to as an execution unit. A General Purpose Register (GPR) file 20 provides registers accessible by the pipeline 12, and comprising the top of the memory hierarchy.

[0019] The processor 10, which executes instructions from at least two instruction sets in different instruction set operating modes, additionally includes a debug circuit 18, operative to compare, upon the execution of each instruction, at least a predetermined target instruction set operating mode to the current instruction set operating mode, and to provide an indication of a match between the two. The debug circuit 18 is described in greater detail below.

[0020] The pipeline 12 fetches instructions from an instruction cache (I-cache) 26, with memory address translation and permissions managed by an Instruction-side Translation Lookaside Buffer (ITLB) 28. Data is accessed from a data cache (D-cache) 30, with memory address translation and permissions managed by a main Translation Lookaside Buffer (TLB) 32. In various embodiments, the ITLB 28 may comprise a copy of part of the TLB 32. Alternatively, the ITLB 28 and TLB 32 may be integrated. Similarly, in various embodiments of the processor 10, the I-cache 26 and D-cache 30 may be integrated, or unified. Misses in the I-cache 26 and/or the D-cache 30 cause an access to main (off-chip) memory 38, 40 by a memory interface 34. The memory interface 34 may be a master input to a bus interconnect 42 implementing a shared bus to one or more memory devices 38, 40. Additional master devices (not shown) may additionally connect to the bus interconnect 42.

[0021] The processor 10 may include an Input/Output (I/O) interface 44, which may be a master device on a peripheral bus, across which the I/O interface 44 may access various peripheral devices 48, 50. Those of skill in the art will recognize that numerous

variations of the processor 10 are possible. For example, the processor 10 may include a second-level (L2) cache for either or both the I and D caches 26, 30. In addition, one or more of the functional blocks depicted in the processor 10 may be omitted from a particular embodiment. Other functional blocks that may reside in the processor 10, such as a JTAG controller, instruction predecoder, branch target address cache, and the like are not germane to a description of the present invention, and are omitted for clarity.

[0022] Figure 2 depicts a functional block diagram of one embodiment of the debug circuit 18. The debug circuit 18 includes a target instruction set operating mode register 52, an address range start address register 54, and an address range end address register 56. The registers 52, 54, 56 are loaded by programmers via diagnostic software prior to a diagnostic/debug operation. The value written to the target instruction set operating mode register 52 is compared to the current processor instruction set operating mode during the execution of each instruction, to trigger a breakpoint, trace function, or other alert. The values of the start and end address registers 54, 56, are the beginning and end, respectively, of the target address range over which the debug circuit 18 is to monitor for a match between the current and target instruction set operating modes.

[0023] As of those of skill in the art will readily recognize, the debug circuit 18 may include additional customizable parameters, and additional functional blocks, enabling breakpoints, traces, and the like to trigger on a wide variety of conditions. These have been omitted from Figure 2 for clarity, but in general may include all debug circuit parameters and functionality known in the art.

[0024] Prior to a diagnostic/debug run, a programmer loads the start and end address registers 54, 56 to define a target address region, which may range from a single address to the entire address range of the code being tested. In other embodiments, multiple start and end address registers may be provided to define a

plurality of target address ranges. The programmer additionally loads the target instruction set operating mode register 52. For example, in the case of an ARM processor executing code entirely in ARM mode, a programmer may load register 52 with a value indicating Thumb mode, to set a breakpoint causing the debug circuit 18 to halt execution, for example, if the processor 10 executes in the instruction within the address range defined by registers 54, 56 in Thumb mode.

[0025] During the diagnostic/debug operation, the value stored in the target instruction set operating mode register 52 is compared to a status register bit indicating the current instruction set operating mode of the processor 10 in comparison logic such as the AND gate 58. For example, in an ARM processor 10, bit 5 of the Current Program Status Register (CPSR) 16 may be monitored. The output 60 of the comparison logic 58 will be asserted only when the current instruction set operating mode of the processor 10 matches the instruction set operating mode indicated by the target instruction set operating mode register 52.

[0026] In embodiments where the processor 10 executes instructions from more than two instruction sets in different instruction set operating modes, the target instruction set operating mode register 52 and mode indicator bits of the CPSR (or other status register) 16 will comprise multi-bit values, and the comparison logic 58 may include additional circuits such as decoders. In various embodiments, other indications of the current instruction set operating mode may be utilized in lieu of the CPSR 16, such as the output of instruction decoding logic.

[0027] Simultaneously, the current value of the PC 15 is compared to the address range registers 54, 56 in address comparison logic 68. The output 70 of the address comparison logic 68 is asserted whenever the current value of the PC 15 falls within the target address range. In various embodiments, indications of the "current" instruction address may vary from the PC 15, as required or desired. For example, the address comparison circuit 68 may compare the address of instructions in a decode

stage of the pipeline 12, an execute stage, or any other pipeline stage. Additionally, a more dynamic address may be used, such as the address of the last instruction in the pipeline 12 to commit to execution.

[0028] The output 70 of the address comparison logic 68 is ANDed with the output 60 of the instruction set operating mode indicator comparison logic 58 at AND gate 72, to generate an output 20 indicating a day match between the current and target instruction set operating modes occurring within the target address range. In one embodiment, the output 20 is provided as an output to the processor (see Figure 1). This output 20 may arm and/or trigger a logic analyzer, illuminate an LED, or trigger some other action or alarm. In one embodiment, as depicted in Figure 1, the instruction set operating mode match indicator output 20 may comprise an input to the controller 14, to halt execution of the processor 10.

[0029] In one embodiment, the instruction set operating mode match indicator output 20 may initialize or terminate an instruction trace operation, with the trace information being directed to output pins of the processor 10 or stored in an on-chip trace buffer (not shown). In one embodiment, as depicted in Figure 2, the instruction set operating mode match indicator output 20 may trigger an instruction set operating mode match address register 74, capturing the value of the PC 15, and hence the address of the instruction at which may match is detected between the current and target instruction set operating modes. This may be of particular value in detecting a change from one instruction set operating mode to another. The output 76 of this register 74 may be routed to pins of the processor 10 for external inspection.

[0030] In one embodiment, the instruction set operating mode match indicator output 20 may cause an exception, branching code execution to a predefined routine. This instruction set mode match routine may, for example, read the instruction set operating mode match address register 74. In general, the instruction set operating

mode match indicator output 20 may trigger any known debugger operation, as required or desired.

[0031] Figure 3 depicts a method of debugging code according to one or more embodiments of the present invention. Starting at block 78, a programmer initializes the debug circuit registers (block 80). This may comprise, for example, loading the target instruction set operating mode register 52, and setting a target address range by loading the start address register 54 and end address register 56. Additional debug circuit registers (not shown) may be initialized, as required or desired.

[0032] The programmer then begins execution of the code to be debugged (block 82), successively fetching and executing the next instruction (block 84) in the code sequence. Regardless of the address, if the current instruction set operating mode of the processor 10 does not match that set by the programmer in the target instruction set operating mode register 52 (block 86), then the next instruction is fetched and executed (block 84) until the end of the code to be debugged (block 88), in which case the method stops at block 90.

[0033] If, upon executing an instruction (block 84), the instruction address is within the predetermined address range, and the current instruction set operating mode matches that set by the programmer in the target instruction set operating mode register 52 (block 86), then the debug circuit asserts output 20 and may take any number of actions, as specified by the programmer.

[0034] For example, the debug circuit output 20 may cause the processor 10 to take an exception, branching code execution to an instruction set operating mode match routine (block 92). The debug circuit output 20 may cause the pipeline controller 14 to halt execution (block 94), allowing the programmer to inspect the contents of various registers, cache lines, and the like. The debug circuit 18 may assert the output 20 externally (block 96), which may be used to arm or trigger a logic analyzer, light an LED, or the like. The debug circuit 18 may additionally output the instruction set

operating mode match address for external inspection (block 98). The debug circuit 18 may start or stop a trace operation (block 100). Although depicted as alternatives, the debug circuit 18 may, in a given embodiment, execute two or more of blocks 92-96. In general, the debug circuit 18 may take any action known in the art to be beneficial to debugging code on the processor 10, having detected a match between the current instruction set operating mode and the target instruction set operating mode.

[0035] As used herein, the term "instruction set" refers to a set of encodings by which instruction data (e.g., the contents of an I-cache line) are interpreted by a processor 10 as executable instructions. The term "instruction set operating mode" refers to an identifiable mode of operation of the processor 10 in which instruction data are interpreted according to a particular instruction set encoding. In particular, "instruction set operating mode" is distinguished from known processor operating "modes" relating to permissions (e.g., supervisor versus user mode), memory addressing (e.g., real versus protected mode), and the like.

[0036] Although the present invention has been described herein with respect to particular features, aspects and embodiments thereof, it will be apparent that numerous variations, modifications, and other embodiments are possible within the broad scope of the present invention, and accordingly, all variations, modifications and embodiments are to be regarded as being within the scope of the invention. The present embodiments are therefore to be construed in all aspects as illustrative and not restrictive and all changes coming within the meaning and equivalency range of the appended claims are intended to be embraced therein.

CLAIMS

What is claimed is:

1. A method of debugging software code comprising a plurality of instructions on a processor having at least two different instruction set operating modes, comprising:
receiving a target instruction set operating mode prior to executing any code instructions;
executing code instructions and, for each instruction executed, comparing the current processor instruction set operating mode to the target instruction set operating mode; and
triggering an alert if the current processor instruction set operating mode matches the target instruction set operating mode.
2. The method of claim 1 wherein the alert comprises halting instruction execution.
3. The method of claim 1 wherein the alert comprises controlling a trace operation.
4. The method of claim 1 wherein the alert comprises causing an exception.
5. The method of claim 1 wherein the alert comprises outputting a signal indicative of a match between the current processor instruction set operating mode and the target instruction set operating mode.
6. The method of claim 1 wherein the alert comprises outputting the address of the instruction for which the current processor instruction set operating mode matches the target instruction set operating mode.

7. The method of claim 1 further comprising:
receiving a target address range prior to executing any code instructions;
for each instruction executed, comparing the current instruction address to the
target address range; and
triggering the alert only if the current processor instruction set operating mode
matches the target instruction set operating mode and the current
instruction address is within the target address range.
8. A method of executing software on a processor having at least first and second
instruction set operating modes, comprising:
identifying the address at which the processor switches from the first instruction
set operating mode to the second instruction set operating mode; and
triggering an alert in response to the identification.
9. The method of claim 8 wherein the alert comprises halting instruction execution.
10. The method of claim 8 wherein the alert comprises controlling a trace operation.
11. The method of claim 8 wherein the alert comprises causing an exception.
12. The method of claim 8 wherein the alert comprises outputting a signal indicative
of a change in instruction set operating mode.
13. The method of claim 8 wherein the alert comprises outputting the address of the
instruction identified.

14. The method of claim 8 wherein identifying the address at which the processor switches from the first instruction set operating mode to the second instruction set operating mode comprises identifying the address only if it falls within a predetermined address range.

15. A processor operative to execute instructions according to two or more instruction set encodings, each in a different instruction set operating mode, comprising:

- a current instruction set operating mode indicator;
- a data storage location operative to store a target instruction set operating mode indicator;
- an execution unit operative to execute instructions according to a current instruction set operating mode; and
- a compare circuit operative to compare the current instruction set operating mode to the target instruction set operating mode upon executing each instruction, and to output an indication if the current instruction set operating mode matches the target instruction set operating mode.

16. The processor of claim 15 wherein the indication comprises a signal operative to halt instruction execution.

17. The processor of claim 15 wherein the indication comprises a signal operative to control a trace operation.

18. The processor of claim 15 wherein the indication comprises a signal operative to cause an exception.

19. The processor of claim 15 wherein the indication comprises a signal indicating that the current instruction set operating mode matches the target instruction set operating mode.

20. The processor of claim 15 wherein the indication comprises the address of the instruction being executed when a match is detected between the current instruction set operating mode and the target instruction set operating mode.

21. The processor of claim 15 further comprising a data storage location operative to store a target address range, and wherein the compare circuit is operative to output an indication only if the current instruction set operating mode matches the target instruction set operating mode and the address of the instruction currently being executed falls within the target address range.

22. A processor operative to execute two or more instruction set encodings, each in a different instruction set operating mode, comprising:

a compare circuit operative to detect a change in the instruction set operating mode and to output an indication of the instruction set operating mode change, and the instruction address at which the change occurs, in response to the detection.

23. The processor of claim 22 wherein the indication comprises a signal operative to halt instruction execution.

24. The processor of claim 22 wherein the indication comprises a signal operative to control a trace operation.

25. The processor of claim 22 wherein the indication comprises a signal operative to cause an exception.

26. The processor of claim 22 wherein the indication comprises a signal indicative of a change in instruction set operating mode.

27. The processor of claim 22 wherein the indication comprises the instruction set operating mode change address.

28. The processor of claim 22 wherein the compare circuit detects a change in the instruction set operating mode only when the instruction address at which the change occurs falls within a predetermined address range.