PROCESSOR IDLE STATE

BACKGROUND

This invention relates to placing a programmable processor in an idle state.

A programmable processor, such as a microprocessor for a computer or a digital signal processing system, may be capable of performing millions of electronic operations per second. In some applications, the processor may have few operations to perform at a given time, yet the processor remains fully powered and ready to perform as needed. The power consumption may be undesirable when, for example, the processor is incorporated in a battery-powered device.

DESCRIPTION OF DRAWINGS

Figure 1 is a block diagram illustrating an example of a pipelined programmable processor.

Figure 2 is a schematic illustrating an example execution pipeline.

Figure 3 is a flowchart illustrating a process for placing a processor in an idle state.

Figure 4 is a logic diagram of an idle handler.

Figure 5 is a flowchart illustrating a process for returning a processor from an idle state.

Figure 6 is a flowchart illustrating a process for returning a processor from an idle state.

DETAILED DESCRIPTION

In some instances, it may be desirable that the operations of the processor be suspended. When the operations of the processor are suspended, power to the processor may be reduced, thus promoting energy efficiency and prolonging battery power. The techniques described below may provide ways of creating an idle mode for a processor, in which the processor performs few or no processing operations, does not fetch instructions for execution, and/or ignores interrupts. Also described are techniques for "waking up" the processor from its idle state and returning it to its active state.

Figure 1 is a block diagram illustrating a programmable processor 10 having an execution pipeline 14 and a control unit 12. Control unit 12 controls the flow of instructions and data through pipeline 14 during a clock cycle. For example, during the processing of an instruction, control unit 12 may direct the various

components of pipeline 14 to decode the instruction and perform the corresponding operation including, for example, writing the results back to memory. Many operations of processor 10 are clocked, with a clock signal supplied by a clock controller 15.

A connection denoted "idle output" 17 couples pipeline 14 to an idle handler 11, shown in Figure 1 as part of control unit 12. Idle handler 11 may bring about the idle state. In addition, idle handler 11 may assert idle output bit 17 to indicate an idle state. Assertion of idle output 17 may stall pipeline 14, and may result in suspension of operations of pipeline 14. Suspension of operations may be part of placing processor 10 in an idle state. Idle handler 11 and idle output 17 are described in more detail below.

In some applications, programmable processor 10 cooperates with a main processor 18. Processor 10 may be a slave or a companion to main processor 18. In other applications, programmable processor 10 may not cooperate with another processor. The system depicted in Figure 1 is intended for illustration and is not intended to limit the scope of the invention. In addition, applications may include additional components not shown in Figure 1,

such as static random access memory, bus controllers, interrupt handlers and input-output devices.

Main processor 18 may sense idle output 17. By sensing idle output 17, main processor 18 detects whether processor 10 is active or idle. Main processor 18 may be further adapted to wake up processor 10 by assertion of a wake-up signal 84.

Instructions, stored in instruction cache memory 16, may be loaded into a first stage of pipeline 14 and processed through the subsequent stages. The stages may process substantially concurrently with the other stages. Data may pass between the stages in pipeline 14 during a clock cycle of the system. The results of the instructions may emerge at the end of pipeline 14 in rapid succession.

Figure 2 illustrates an example pipeline 14. Pipeline 14, for example, may have five stages: instruction fetch (IF), instruction decode (DEC), address calculation (AC), execute (EX) and write back (WB). Instructions may be fetched from memory, or from instruction cache 16, during the first stage (IF) by fetch unit 20 and decoded 22 within address registers 24 during the second stage (DEC). At the next clock cycle, the results may be passed to the third stage (AC), where

data address generators 26 calculate any memory addresses that are used to perform the operation. During the execution stage (EX), execution unit 28 may perform the operation such as, for example, adding or multiplying two numbers. During the final stage (WB), the results may be written back to data memory or to data registers 30.

An application including programmable processor 10 may encounter situations in which processor 10 has no tasks to perform, yet is consuming power. In such situations it may be advantageous for processor 10 to enter a low-power idle mode. An idle mode may be initiated by, for example, a user that instructs processor 10 to enter a low-power state, or by main processor 18 after a period has elapsed in which processor 10 has had no tasks to perform, although the scope of the invention is not limited in this respect. Figure 3 illustrates a process by which processor 10 in an active state (50) may enter a low-power idle mode. When an idle is indicated (52), such as by a user or by main processor 18, processor 10 may disable interrupts (54). One way to disable interrupts is to execute a CLI (clear interrupt) instruction that alters the interrupt mask used by the interrupt handler. As a consequence of executing the CLI instruction, processor 10 may ignore

interrupts while entering the idle state. Processor 10 may also ignore interrupts while in the idle state.

Processor 10 then places itself in a known idle state (56). Placing processor 10 in a known idle state allows processor 10 to exit from the idle mode at a later time in a known state, without undergoing a reset. Figure 3 shows one way to place processor 10 in an idle state (56). An idle request may be indicated (52) by sending an idle instruction through pipeline 14. As will be described below, the idle instruction may result in the setting of an idle flag bit 80 (see Figure 4) (58). Idle flag bit 80 may indicate that processor 10 is entering an idle state. Idle flag bit 80 may be stored in a memory element such as a register 102 (see Figure 4).

The idle instruction may be followed in pipeline 14 by a system synchronization (SSYNC) instruction (60). In general, an SSYNC instruction may specify that all pending processing operations must be completed before the next operation is initiated. Accordingly, the SSYNC instruction may allow instructions already in pipeline 14 to emerge, but may stall pipeline 14 from fetching a new instruction from instruction cache 16 until a system acknowledgement 82 (see Figures 1 and 4) is received.

System acknowledgment 82 may indicate that all pending system operations have been completed. In other words, sending SSYNC through pipeline 14 may stall pipeline 14 behind it, resulting in the inactivation of IF stage, DEC stage, and AC stage, while allowing pending instructions ahead of it to complete execution. For example, an instruction ahead of the SSYNC instruction that makes a request to a system bus is not stalled, but rather is allowed to complete execution, thus preserving bus protocol. SSYNC may stall in the EX stage until the system acknowledges the SSYNC instruction.

The SSYNC instruction may perform operations other than in connection with placing processor 10 in an idle state, but when executed in combination with the idle instruction that sets the idle flag bit 80 (58), the SSYNC instruction brings about an idle state as will be shown below. The SSYNC instruction may generate a known state in processor 10, because the SSYNC instruction stalls pipeline 14 until all operations prior to the SSYNC instruction are completed. In addition, the SSYNC instruction may "kill," or cancel, all or some operations in pipeline 14 when the SSYNC instruction reaches the WB stage.

The system acknowledgement signal 82 may be used to indicate that instructions preceding the SSYNC instruction have been executed by processor 10 and by any other components of the system, such as main processor 18 and bus controllers. Upon receipt by processor 10 of a system acknowledgment signal 82, the SSYNC instruction would ordinarily enter the WB stage and complete execution. Because the idle instruction has set idle flag bit 80 (58), however, pipeline 14 may remain stalled and the SSYNC instruction may remain in the EX stage. System acknowledgement 82 may be received by idle handler 11 (62). In addition, the state of idle flag bit 80 may be sensed (64), resulting in the assertion of an idle output signal (66) at idle output 17. Pipeline 14 may remain stalled as long as idle output 17 is asserted. Because pipeline 14 is stalled, the SSYNC instruction may be stalled in the EX stage and may not enter the WB stage.

Idle handler 11 may clear idle flag bit (68). Detection of idle flag bit 80 (64), setting idle output 17 (66) and clearing idle flag bit 80 (68) may be performed by logic in idle handler 11 as described below. In addition, main processor 18 may sense idle output 17, and consequently may sense that processor 10 is in an

idle state, and turns off the clocks to processor 10 (70). In addition to turning off the clocks, main processor 18 may reduce the power supplied to processor 10 (70), e.g., by setting the voltage supply to an "idle mode voltage level." The idle mode voltage level generally would less than the "active mode voltage level" at which processor 10 normally operates, but sufficient to retain the state in the registers. Because power consumption is generally proportional to the square of voltage, reduction from an active mode voltage level in particular embodiments, such as 1.3 volts, to an idle mode voltage level, such as 0.7 volts, may result in a considerable power saving. With clocks turned off and power supply reduced, processor 10 is in a low-power idle mode.

Figure 4 is a logic diagram of an idle handler 11 in accordance with an embodiment of the present invention. Figure 4 illustrates techniques for carrying out the method described above. While processor 10 is active, all bits in the diagram are deasserted. Idle flag bit 80 may not be set and idle output pin 17 may not be asserted. When the idle instruction enters the WB stage, a bit is asserted at reference numeral 96. This bit may be latched by a storage element such as register 102. On

the next clock cycle, register 102 may set idle flag bit 80. Idle flag bit 80 may remain asserted because of feedback to OR gate 98 until idle output 17 is asserted.

When idle flag bit 80 is asserted AND (88) an acknowledgement to SSYNC 82 is received, a high bit may be generated, which is latched in register 94. On the next clock cycle, idle output 17 may be set. The high idle output 17 may be inverted and passed to AND gate 100, which places a deasserted bit in register 102, resulting in idle flag bit 80 being cleared on the following clock cycle. Idle output 17 may remain high because of feedback to OR gate 90 until a wake-up signal 84 is asserted. Until wake-up signal 84 is asserted, processor 10 may remain idle.

A method embodiment for returning the processor to an active state is illustrated in Figure 5. Main processor 18 may restore clocks to processor 10 and may restore the power supply to an operating level, such as by setting the power supply to the active mode voltage level (110). Main processor 18 may also generate a wake-up signal 84 (112). As shown in Figure 4, wake-up signal 84, inverted and passed to AND gate 92, deasserts a bit that may be latched in register 94, clearing idle output bit 17 on the following clock cycle (114).

Clearing idle output 17 may free pipeline 14 from being stalled. Because a system acknowledgment 82 has been sent, the SSYNC instruction enters the WB stage, canceling all operations in the pipeline 14 (116). At this point the flow of instructions into pipeline 14 begins again, by fetching the instruction in instruction cache 16 that follows the SSYNC instruction (118). The instruction following SSYNC is typically an STI (restore interrupt) instruction, which restores the interrupt mask (120). Pipeline 14 may then process instructions in program order (122), meaning that upon waking up, processor 10 may resume where it left off when processor 10 entered idle mode.

While in idle mode, processor 10 may ignore interrupts. Processor 10 may also ignore interrupts received prior to clock restoration (110). After clock restoration, interrupts may be captured but may not be acted upon until interrupt restoration (120). After interrupt restoration (120), interrupts may be processed by the interrupt handler of processor 10 in the ordinary manner.

An alternate embodiment of returning the processor to an active state is illustrated in Figure 6. Figure 6 is like Figure 5, except that main processor 18 generates

an interrupt (126). The interrupt may occur at any stage shown in Figure 6 after clocks are restored (110) and before the interrupt mask is restored (120). As noted above, the interrupt may be captured but not acted upon until interrupt restoration (120), when control is transferred by the interrupt handler to an interrupt service routine (124). As a result, processor 10 may execute the instructions of the interrupt service routine (124) instead of processing instructions in program order.

A number of embodiments of the invention have been described. These and other embodiments are within the scope of the following claims.

CLAIMS

1. A method comprising:

disabling interrupts to a processor;

placing the processor in an idle state; and

asserting a signal at an output terminal indicative of the idle state.

2. The method of claim 1 further comprising disabling clock signal inputs to the processor.

3. The method of claim 1 further comprising reducing the power supply to the processor.

4. The method of claim 1 further comprising:

setting an idle flag;

processing a system synchronization instruction, the system synchronization instruction resulting in generation of an acknowledgment signal; and

placing the processor in an idle state upon sensing of the idle flag and the acknowledgement signal.

5. The method of claim 4 wherein processing the system synchronization instruction results in the processor

entering a known state before the acknowledgement signal is generated.

6. The method of claim 1 wherein the processor includes a pipeline, the method further comprising canceling all operations in the pipeline.

7. The method of claim 1 further comprising receiving a wake-up signal.

8. A device comprising:

an execution pipeline; and

an idle handler coupled to the execution pipeline;

wherein the idle handler is adapted to stall the execution pipeline in response to an idle instruction in conjunction with a system synchronization acknowledgement.

9. The device of claim 8, the idle handler comprising an output terminal, wherein the idle handler is adapted to assert a signal at the output terminal in response to the idle instruction in conjunction with the system synchronization acknowledgement.

10. The device of claim 9, wherein the output terminal is coupled to a memory element.

11. The device of claim 8, the idle handler comprising a wake-up input terminal, the idle handler adapted to discontinue stalling the execution pipeline in response to a signal received on the wake-up input terminal.

12. The device of claim 8, wherein the idle handler is adapted to set an idle flag bit in response to the idle instruction and to clear the idle flag bit in response to a system synchronization acknowledgement.

13. A system comprising:

a first processor, the first processor adapted to enter an idle state and supply an idle output signal;

a second processor coupled to the first processor;

a clock adapted to supply a clock signal to the first processor; and

static random access memory coupled to the first processor;

wherein the second processor deactivates the clock signal supplied to the first processor in response to sensing the idle output signal.

14. The system of claim 13 wherein the first processor comprises an execution pipeline and an idle handler coupled to the execution pipeline.

15. The system of claim 13, wherein the first processor is adapted to ignore interrupts issued from the second processor while in the idle state.

16. The system of claim 13, wherein the second processor is adapted to supply a wake-up signal to the first processor, and the first processor is adapted to exit the idle state when the first processor senses the wake-up signal.

17. A method comprising:

supplying clock signals to a processor, wherein the processor is in an idle state and is not responsive to interrupts;

signaling the processor to exit the idle state; and

restoring the responsiveness of the processor to interrupts.

18. The method of claim 17 further comprising supplying power to the processor at a normal operating level.

19. The method of claim 17 further comprising:

supplying an interrupt to the processor; and

transferring control to an interrupt service routine after responsiveness of the processor to interrupts has been restored.

20. The method of claim 17, wherein the processor includes an execution pipeline, the method further comprising canceling all operations in the execution pipeline.

21. The method of claim 20, further comprising fetching an instruction after the cancellation.

22. The method of claim 21, wherein the fetched instruction restores the responsiveness of the processor to interrupts.

23. The method of claim 17, wherein the processor supplies an idle output signal, the method further comprising clearing the idle output signal.

24. A method, comprising:

operating a processor in a first mode;

disabling interrupts to the processor; and

placing the processor in a second mode,

wherein the processor consumes less power in the second mode than in the first mode.

25. The method of claim 24 further comprising disabling clock signal inputs to the processor.

26. The method of claim 24 further comprising reducing the power supply to the processor.

27. The method of claim 24 wherein the processor includes an execution pipeline, and wherein the execution pipeline performs no operations when the processor is in the second mode.

28. The method of claim 24 further comprising issuing an output signal when the processor is in the second mode.

29. The method of claim 24 further comprising:

issuing an input signal to the processor when the processor is in the second mode;

returning the processor to the first mode; and

enabling interrupts to the processor.

30. The method of claim 29, wherein the processor includes an execution pipeline, returning the processor to the first mode further comprising canceling all operations in the execution pipeline.