

SECURITY ON HARDWARE LOOPS

BACKGROUND

In designing a programmable processor, such as a digital signal processing (DSP) system, processing speed and power consumption may form tradeoffs. Conventional processors include a variety of hardware designed to increase the speed at which software instructions are executed. The additional hardware, however, typically increases the power consumption of the processor.

A "hardware loop" may increase the speed of a programmable processor. This may be implemented using dedicated hardware designed to expedite the execution of software instructions within a loop construct. Hardware loops may reduce the number of clock cycles used to execute a software loop by caching the instructions in local registers, thereby reducing the need to fetch the same instruction from a memory device or instruction cache a number of times.

Hardware loops introduce several challenges. These challenges include avoiding penalties such as setup penalties or branch penalties. Setup penalties include the loss of performance (usually an increase in processing

time) associated with setting up a hardware loop.

Similarly, branch penalties are the loss of performance (again, usually an increase in processing time) associated with a branch.

5

BRIEF DESCRIPTION OF DRAWINGS

Figure 1 is a block diagram illustrating an example of a pipelined programmable processor according to an embodiment of the invention.

10 Figure 2 is a block diagram illustrating an example execution pipeline for the programmable processor in accordance with an embodiment of the present invention.

Figure 3 is block diagram illustrating hardware loop control architecture according to an embodiment.

15 Figure 4 is a flow diagram illustrating an operation for providing security on a hardware loop according to an embodiment.

Figure 5 is a block diagram of a mobile video unit including a processor according to an embodiment.

20

DETAILED DESCRIPTION

Figure 1 is a block diagram illustrating a programmable processor 100 arranged to support hardware loops and security on hardware loop operations without
5 significantly increasing power consumption.

The processor 100 may be operable in a supervisor mode and in a user mode. The processor 100 may operate in the supervisor mode when used by a supervisor program such as an operating system. The processor 100 may operate in the
10 user mode when used by a user program such as an application.

In the supervisor mode, the processor 100 may have certain privileges. These privileges may include the use of certain instructions, access to certain resources, e.g.,
15 registers, and extended read/write access to memory. These privileges may enable the supervisor program to control the overall state of the system, for example, by putting the system to "sleep", writing to architectural registers and performing read/write operations to memory locations used
20 for vital system operations.

In the user mode, the processor 100 may not have such privileges. This restriction may be provided as a security

measure, preventing the user program from accidentally or intentionally influencing system level operations. For example, a malicious user program, such as a virus, could corrupt or delete operating system files or otherwise

5 disable the system with the appropriate access privileges.

The processor 100 may use hardware loops in the user mode. An implementation of a hardware loop operation may provide a user program a backdoor into the supervisor mode, thereby creating a security hazard. In an embodiment, the

10 hardware loop operation includes a security feature, which may be implemented in hardware that removes this security hazard.

In order to support hardware loops, the processor 100 may support a loop setup instruction that initializes the

15 hardware by setting entry and exit conditions for the loop. Entry and exit conditions may be defined by loop conditions: top, bottom and count. The top condition defines the first instruction (or top) of a loop. The bottom condition defines the last instruction (or bottom)

20 of a loop. The count condition defines the number of iterations of the loop.

An instruction in the program code being executed by the processor 100 may have an associated instruction

address, or program count (PC), which points to that instruction's location in memory. In an embodiment, instructions may be aligned at even addresses, for example, the instruction addresses may be two bytes (16-bit) or four
5 bytes (32-bit) wide.

Program code may be executed sequentially unless an event occurs, such as a taken branch or an interrupt, that causes the program counter to branch to a different location. Thus, absent a change in program flow, the PC of
10 a subsequent instruction is the address of the subsequent instruction in the instruction address space. For example, if the PC of the current instruction is 2002 (in hexadecimal) and the instruction is 16-bits (two bytes) wide, the PC of the subsequent instruction would be 2004.

15 Entry of a hardware loop may occur at the first "top match." A top match may occur when the PC is pointing to the top instruction of a loop. Exit of a hardware loop may occur at the last "bottom match." A bottom match may occur when the PC is pointing to the bottom instruction of a
20 loop.

By initializing the count at the first top match and decrementing the count at each bottom match, the hardware may keep track of when it has encountered the last bottom

match. In this manner, the loop conditions top, bottom and count may define the entry and exit conditions of a hardware loop.

The processor 100 may include an execution the
5 pipeline 102 and the control unit 104. The control unit 104 may control the flow of instructions and/or data through the pipeline 102 during a clock cycle. For example, during the processing of an instruction, the control unit 104 may direct the various components of the
10 pipeline to decode the instruction and correctly perform the corresponding operation including, for example, writing the results back to memory.

Instructions may be loaded into a first stage of the pipeline 102 and processed through subsequent stages. A
15 stage may process concurrently with the other stages. Data may pass between the stages in the pipeline 102 during a cycle of the system. The results of an instruction may emerge at the end of the pipeline 102 in rapid succession.

The control unit 104 may include hardware loop unit
20 108 that, as described below, may facilitate fast hardware loops without significantly increasing power consumption of the processor 100.

Figure 2 is a block diagram illustrating an example pipeline 200. The pipeline 200 has multiple stages that may facilitate execution of multiple instructions during a single clock cycle. In pipeline 200, an instruction may enter the instruction fetch (IF) stage 202 during a first clock cycle. The instruction may then continue down the pipeline during subsequent clock cycles. Another instruction may enter the IF stage 202 after a previous instruction has exited. Thus, typically another instruction enters the IF stage 202 during a subsequent clock cycle and then continues down the pipeline during subsequent clock cycles. Similarly, additional instructions enter the IF stage 202 during subsequent clock cycles respectively. The number of stages in the pipeline may define the number of instructions that the pipeline may service simultaneously.

The different stages of the pipeline may operate as follows. Instructions may be fetched during the IF stage 202 by a fetch unit 204 and decoded from instruction registers 206 during a decode (DEC) stage 208. During an address calculation (AC) stage 210, data address generators 212 may calculate any memory addresses used to perform the operation.

During the execution stages (EX 1 to EX n) 214 and 220, execution units 222 and 224 may perform specified operations such as, for example, adding or multiplying two numbers. Execution units may contain specialized hardware
5 for performing the operations including, for example, one or more arithmetic logic units (ALUs), floating-point units (FPU) and barrel shifters.

A variety of data may be applied to the execution units such as the addresses generated by data address
10 generators, data retrieved from memory or data retrieved from data registers. During a write back stage (WB) 230, the results may be written to a memory location or data registers external to the pipeline or to data registers in the pipeline such as architectural registers 232. The
15 stages of pipeline 200 may include one or more storage circuits, such as a flip-flop, for storing data.

Figure 3 is a block diagram of the loop unit 108 of Figure 1. The hardware loop unit 108 may include a LOOP_TOP register 302, a LOOP_BOT register 304, and
20 LOOP_CNT register 306, and a loop control unit 312. The hardware loop unit may initialize a loop by loading the address of the top instruction of the loop in the LOOP_TOP register 302, the address of the bottom instruction in the

loop in the LOOP_BOT register 304, and a count indicating the number of times the loop is to be completed in the LOOP_CNT register 306. The loop control unit may compare the values in the LOOP_TOP register 302 and the LOOP_BOT
5 register 304 to the current PC. Upon detecting a top match, the loop control unit may begin a hardware loop operation. When a bottom match occurs, the loop control unit 312 may decrement the value in the LOOP_CNT register 306 and branch the PC back to the address of the top
10 instruction. The hardware loop may operate in the pipeline until the exit condition of the loop has been satisfied ,e.g., a bottom match with count equal to zero.

A security hazard may exist if the user program obtains an address of an instruction in the supervisor
15 program address space, for example, an instruction address in an event vector table (EVT) 120 (Figure 1). The EVT may include the target address for event service routines. Events may include exceptional conditions that cause a break in program flow. Events such as I/O device
20 interrupts, misaligned memory accesses, hardware malfunctions, etc., may be handled in the processor 100 by a corresponding event service routine. When a particular event occurs, the processor 100 may look up the appropriate

target address for the event service routine for that event in the EVT 120 and branch the PC to the event service routine to handle the event. When the processor 100 services an event, it enters the supervisor mode and gains
5 the corresponding privileges. After the event is handled, the processor 100 may return to the user mode and continue the interrupted program flow.

A user program that acquired the a target address in the EVT 120, for example, for an event service routine for
10 handling a common I/O device interrupt, could load the LOOP_BOT register 304 with the address of an event service routine and load the LOOP_TOP register 302 with the address of an instruction in the user program address space.

Consider a situation in which the supervisor code is
15 located at hexadecimal addresses 1000 to 2000 and the user code is located at hexadecimal addresses 10,000 to 20,000. In this example, the target address for the event service routine for the I/O device interrupt is instruction address 1004 and the user program loads 1004 into the LOOP_BOT
20 register and loads 10,000 into the LOOP_TOP register. When that I/O device interrupt occurs, the supervisor program checks the EVT 120 and branches the PC to instruction 1004. This causes a bottom match. In response to the bottom

match, the hardware loop unit 108 causes the PC to implicitly branch to the instruction address in the LOOP_TOP register 302, 10,200, which resides in the user code address space. When the PC branches to the

5 instruction in the user code, the user program gains control of the processor. However, the processor 100 may remain in the supervisor mode. Hence the user program acquires supervisory privileges it is usually denied. A malicious user program could use those privileges to

10 disrupt or disable the system.

Figure 4 is a flowchart of an operation 400 according to an embodiment. The following description is one embodiment of implementing the operation 400. In other embodiments, blocks may be skipped or performed in a

15 different order.

As shown in Figure 3, a mode detector 320 may be coupled to the LOOP_BOT register 304. The mode detector 320 may detect when the processor exits the user mode in block 402. When the processor 100 exits the user mode, the

20 mode detector 320 may set the least significant bit (LSB) of the instruction address in the LOOP_BOT register to one in block 404.

Since the instruction addresses are aligned on even addresses, the loop control unit 310 should not encounter an odd address; the LSB should always be zero. Hence, a bottom match should not occur when the processor is in the
5 supervisor mode. Consider the example given above. The user program loads instruction address 1004 (binary - "0000 0011 1110 1100") in the LOOP_BOT register 304 in the user mode. When the processor exits the user mode, the mode detector 320 sets the LSB of the address to one, and the
10 value in the LOOP_BOT register becomes 1005 ("0000 0011 1110 1101").

Since all instruction addresses should be evenly aligned, if the PC does encounter an odd address in the program flow in block 406, the processor 100 may take an
15 exception in block 408. In an embodiment, the exception has a higher priority than a bottom match. Thus, even though a bottom match occurred, the processor 100 branches to the event service routine for handling the exception rather than branching to the address in the LOOP_TOP
20 register 302, thereby avoiding the security hazard.

When the processor reenters the user mode in block 410, the mode detector 320 may set the LSB of the value in

the LOOP_BOT register 306 back to zero in block 412, thereby re-enabling the hardware loop architecture.

Such a processor 100 is contemplated for use in video camcorders, teleconferencing, PC video cards, and High-
5 Definition Television (HDTV). In addition, the processor 100 is also contemplated for use in connection with other technologies utilizing digital signal processing such as voice processing used in mobile telephony, speech
recognition, and other applications.

10 For example, Figure 5 illustrates a mobile video device 500 including a processor 100 according to an embodiment. The mobile video device 500 may be a hand-held device which displays video images produced from an encoded video signal received from an antenna 502 or a digital
15 video storage medium 504, e.g., a digital video disc (DVD) or a memory card. The processor 100 may communicate with a cache memory 506, which may store instructions and data for the processor operations, and other devices, for example, an SRAM 508.

20 The processor 100 may be a microprocessor, a digital signal processor (DSP), a microprocessor controlling a slave DSP, or a processor with an hybrid microprocessor/DSP architecture. The processor 100 may perform various

operations on the encoded video signal, including, for example, analog-to-digital conversion, demodulation, filtering, data recovery, and decoding. The processor 100 may decode the compressed digital video signal according to
5 one of various digital video compression standards such as the MPEG-family of standards and the H.263 standard. The decoded video signal may then be input to a display driver 510 to produce the video image on a display 512.

A number of embodiments of the invention have been
10 described. Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.

CLAIMS:

1 1. A method comprising:
2 operating a processor in at least first and
3 second operating modes; and
4 disabling a hardware loop operation associated
5 with the processor upon exiting said first operating mode.

1 2. The method of claim 1, wherein the first mode is
2 a user mode.

1 3. The method of claim 2, wherein the second mode is
2 a supervisor mode.

1 4. The method of claim 1, further comprising:
2 initializing the hardware loop operation in the
3 first mode; and
4 detecting an exit from the first mode.

1 5. The method of claim 4, wherein said initializing
2 the hardware loop operation comprises:
3 loading an evenly aligned top instruction address
4 in a first register; and

5 loading an evenly aligned bottom instruction
6 address in a second register.

1 6. The method of claim 5, wherein said disabling the
2 hardware loop operation comprises setting a least
3 significant bit of the bottom instruction address in the
4 second register to one.

1 7. The method of claim 5, wherein the top and bottom
2 instruction addresses comprise evenly aligned byte
3 addresses.

1 8. The method of claim 1, further comprising:
2 enabling a hardware loop operation upon entering
3 the first operating mode.

1 9. The method of claim 8, further comprising:
2 loading an evenly aligned top instruction address
3 in a first register;
4 loading an evenly aligned bottom instruction
5 address in a second register;
6 detecting entry into the first operating mode;
7 and

8 setting a least significant bit of the bottom
9 instruction address in the second register to zero.

1 10. The method of claim 8, wherein the top and bottom
2 instruction addresses comprise evenly aligned byte
3 addresses.

1 11. An article comprising:
2 a machine-readable medium which stores machine-
3 executable instructions for providing security in a
4 processor, the instructions causing a machine to:
5 operate the processor in at least first and
6 second operating modes; and
7 disable a hardware loop operation upon exiting a
8 first operating mode.

1 12. The article of claim 11, wherein the first mode
2 is a user mode.

1 13. The article of claim 12, wherein the second mode
2 is a supervisor mode.

1 14. The article of claim 11, further comprising
2 instructions that cause the machine to:

3 initialize the hardware loop operation in the
4 first mode; and
5 detect an exit from the first mode.

1 15. The article of claim 14, wherein the instructions
2 that cause the machine to initialize the hardware loop
3 operation comprise instructions causing the machine to:
4 load an evenly aligned top instruction address in
5 a first register; and
6 load an evenly aligned bottom instruction address
7 in a second register.

1 16. The article of claim 15, wherein the instructions
2 that cause the machine to disable the hardware loop
3 operation comprise instructions that cause the machine to
4 set a least significant bit of the bottom instruction
5 address in the second register to one.

1 17. The article of claim 15, wherein the top and
2 bottom instruction addresses comprise evenly aligned byte
3 addresses.

1 18. The article of claim 11, further comprising
2 instructions that cause the machine to:

3 enable a hardware loop operation upon entering
4 the first operating mode.

1 19. The article of claim 18, further comprising
2 instructions that cause the machine to:

3 load an evenly aligned top instruction address in
4 a first register;

5 load an evenly aligned bottom instruction address
6 in a second register;

7 detect entry into the first operating mode; and
8 set a least significant bit of the bottom instruction
9 address in the second register to zero.

1 20. The method of claim 18, wherein the top and
2 bottom instruction addresses comprise evenly aligned byte
3 addresses.

1 21. A processor comprising:

2 a first register to store an address of a top
3 instruction in a hardware loop;

4 a second register to store an address of a bottom
5 instruction in the hardware loop;

6 a hardware loop controller coupled to the first
7 and second registers, said controller operative to perform
8 the hardware loop operation; and

9 a mode detector operative to disable the hardware
10 loop controller in response to the processor exiting the
11 first operating mode.

1 22. The processor of claim 21, wherein the first
2 operating mode is a user mode.

1 23. The processor of claim 22, wherein the processor
2 is further operable in a supervisor mode.

1 24. The processor of claim 21, wherein the mode
2 detector is operative to enable the hardware loop
3 controller in response the processor entering the first
4 operating mode.

1 25. The processor of claim 21, wherein the address of
2 the bottom instruction is evenly aligned.

1 26. The processor of claim 25, wherein the top and
2 bottom instruction addresses comprise evenly aligned byte
3 addresses.

1 27. The processor of claim 25, wherein the mode
2 controller is operative to set a least significant bit of
3 the address in the second register to one in response to
4 the processor exiting the first mode.

1 28. The processor of claim 25, wherein the mode
2 controller is operative to set a least significant bit of
3 the address in the second register to zero in response to
4 the processor entering the first mode.

1 29. A system comprising:
2 a static random address memory; and
3 a processor coupled to the static random access
4 memory, said processor comprising:
5 a first register to store an address of a top
6 instruction in a hardware loop;
7 a second register to store an address of a bottom
8 instruction in the hardware loop;
9 a hardware loop controller coupled to the first
10 and second registers, said controller operative to perform
11 the hardware loop operation; and

12 a mode detector operative to disable the hardware
13 loop controller in response to the processor exiting a
14 first operating mode.

1 30. The processor of claim 29, wherein the first
2 operating mode is a user mode.

1 31. The processor of claim 30, wherein the processor
2 is further operable in a supervisor mode.

1 32. The processor of claim 29, wherein the mode
2 detector is operative to enable the hardware loop
3 controller in response the processor entering the first
4 operating mode.