MOTOROLA
SEMICONDUCTOR
ENGINEERING BULLETIN

Order this document
by EB164/D

EB164

Interrupt Latency in the MC88110

This document addresses the interrupt latency in the MC88110. It provides a brief description of how interrupts are handled in the MC88110. It also includes examples of short and long interrupt latency cases. This document is intended for those hardware system designers who have read and are familiar with the *MC88110 Second Generation RISC Microprocessor User's Manual.*

NOTE

This document does not use the terms "exception" and "interrupt" interchangeably. The term **exception** is used to describe any event, other than a branch or jump, that changes the normal flow of instruction execution. The term **interrupt** is used to define any external exception signaled on the INT or NMI pins of the MC88110 and is the focus of this document.

Interrupts in the MC88110

Interrupts in the MC88110 are a specific type of external exception and are present in two forms: maskable and nonmaskable. The maskable interrupt is generated when the INT input signal is asserted. The nonmaskable interrupt is generated when the NMI input signal is asserted.

The NMI signal is transition sensitive (falling edge) and must be held asserted until it is acknowledged by the interrupt handler. Once an NMI signal is recognized by the MC88110, NMI must be negated and reasserted before another nonmaskable interrupt can be recognized.

The INT signal is level sensitive, so the external hardware must keep the signal asserted until the interrupt is recognized. This recognition is normally generated by an interrupt handler.

If the interrupt disable (IND) bit in the processor status register (PSR) is set, all external maskable interrupts (signaled by INT) are disabled. If the exceptions freeze (EFRZ) bit in the PSR is set, then exception-time registers are frozen and both types of interrupts are disabled. When the EFRZ bit is set, any recognized exception (except interrupts) will be directed to the error exception handler. A maskable or nonmaskable interrupt will never be directed to the error exception handler since the EFRZ bit also disables INT and NMI.

Table 1 illustrates whether an interrupt or exception will be recognized or masked according to which bits (IND, EFRZ) are cleared. The column labeled Other Exceptions describes what happens if a bus or internal exception occurs.

MOTOROLA

9/92

IND	EFRZ	INT	NMI	Other Exceptions
0	1	Disabled	Disabled	Error exception handler
1	1	Disabled	Disabled	Error exception handler
1	0	Disabled	Goes to its exception handler	Goes to its exception handler
0	0	goes to its exception handler	Goes to its exception handler	Goes to its exception handler

Table 1. Interrupt Masking

Interrupt Latency

Interrupt latency is affected by two primary factors—the time from when the INT signal asserts until it is recognized by the processor (exception recognition); and the time from when the processor recognizes the interrupt until the interrupt handler begins execution (exception processing). These measurements of time are dependent on the state of the processor when the interrupt occurs. Examples of these dependencies are given in the following sections.

The MC88110 maintains a precise exception model. When an interrupt signal asserts, the precise address of the faulting instruction is provided to the exception handler and the MC88110 is restored to a state such that neither the faulting instruction or any instructions that follow appear to have executed. This restoration process is facilitated by the history buffer that keeps track of the order of issuance of each instruction.

EXCEPTION RECOGNITION

The time from when an interrupt signal asserts to when the processor recognizes it is dependent upon three factors:

•The number of instructions that are in the history buffer at the time of the interrupt,

•Whether a memory access instruction such as a load, store or **xmem** in the history buffer had already accessed the cache or bus when the interrupt signal was asserted,

•Latency of the instruction at the top of the history buffer. The MC88110 is designed so that the interrupt receives recognition as quickly as possible.

Typically, the MC88110 initiates recognition of an interrupt as soon as an interrupt is detected. In the MC88110, the NMI is detected by the processor three cycles after it asserts, and the INT is detected two cycles after it asserts.

The MC88110 initiates the process of interrupt recognition by first stalling instruction issue, disabling the **ld/st** unit, and marking any pending loads or stores in the **ld/st** unit that have not touched the cache or the bus, as causing an exception. Additionally, any instruction in the history buffer that finishes execution and performs a register write-back is also tagged with an exception. The only instructions that do not write-back are branches (except **bsr**) and stores. If the history buffer is empty, the first instruction, that enters the buffer after the interrupt is detected, is marked as causing an unimplemented opcode exception. The external interrupt is recognized when the first instruction that is tagged with an exception reaches the top of the history buffer. If the instruction already at the top of the buffer has not yet performed a write-back when

the interrupt is detected, it is allowed to finish executing and the interrupt is recognized the cycle after the instruction completes write-back.

It will take longer to recognize an interrupt if a load, store or **xmem** has already accessed the bus or cache at the time the interrupt signal is detected. Memory access instructions that have accessed the bus or cache are allowed to finish execution before the process of interrupt recognition is allowed to start. Execution time of load and store operations will depend on whether the data is available in the cache or an access to external memory is necessary. The memory access latency will determine how quickly the load or store will complete in the case of a cache miss.

If a memory instruction such as a load has accessed the cache or bus, and an interrupt is detected, the MC88110 waits until the load writes back, before initiating the process leading to interrupt recognition. After the load writes back, the MC88110 proceeds with the process of interrupt recognition as described in the previous paragraphs.

Incorrectly predicted branches that are conditional upon immediately preceding loads may further delay an interrupt from being recognized. If the branch reaches the head of the buffer after the load writes back and the processor determines that it incorrectly predicted the branch, not only will the processor need to back out of all of the speculatively issued instructions in the buffer, but it will also have to fetch the first instruction from the correct branch target. This may take many clocks depending on the instruction memory access time. Once the first instruction in the correct branch path is fetched, it is tagged with an unimplemented opcode exception. The interrupt is recognized when this instruction reaches the top of the history buffer.

EXCEPTION PROCESSING

Execution of the interrupt handler begins with the execution of the first instruction in the exception vector. Before the interrupt handler can begin, the MC88110 needs to attain a precise machine state, consistent with a sequential order of program flow. When the processor recognizes the interrupt, it restores the processor to the machine state that existed at the time the excepting instruction (that is, the instruction at the top of the buffer when the interrupt is recognized) was issued. The number of instructions occupying the history buffer when the interrupt is recognized affects the time it takes the processor to restore the correct machine state. The MC88110 restores at two instructions per clock cycle and takes a maximum of six clocks to restore the processor to its initial machine state, since a maximum of 11 additional instructions could have been issued when the interrupt was asserted.

Once the processor has been restored to its correct machine state, the MC88110 performs all the actions necessary to transfer control to the exception handler. This process takes three clocks plus whatever amount of time is required to fetch the first handler instruction from cache or memory, as appropriate.

Interrupt Latency Formula

All of the factors listed previously make up the interrupt latency for the MC88110. For this section, interrupt latency is defined as the time from when the interrupt signal asserts until the instruction that returns the processor to normal execution has been issued. Therefore, the following interrupt latency equation includes all four factors listed previously. In addition, the time for detection is also included. Detection is the time from when the interrupt asserts to when it is detected by the MC88110.

Interrupt latency formula = detection + recognition + processing + handling + return time

detection	= 2 cycles for INT or 3 cycles for NMI
recognition	= dependent on the sequence of instructions in the history buffer
processing	= instruct + 3 + mwait

where:

instruct = [#/2] where # = the number of instructions in the history buffer when the interrupt is recognized

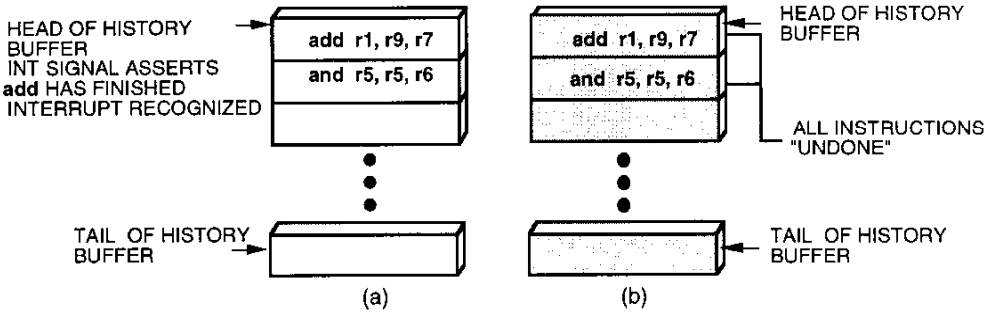
mwait = the number of cycles needed to fetch an instruction from memory.

The recognition time is completely dependent on the sequence of instructions in the history buffer at the time the interrupt is signaled. No formula is given for this portion of the interrupt latency time, instead, the next section discusses some scenarios that result in short and long interrupt latencies.

SHORT INTERRUPT RECOGNITION LATENCY

Often, an interrupt can be detected and recognized one clock cycle after it is received. If no memory instructions are in the history buffer, the MC88110 will detect the interrupt as soon as it is received. The interrupt will be recognized when the first instruction that has completed a write-back, after detection, reaches the top of the history buffer.

Example 1: The first case, illustrated in Figure 1, is an example of an interrupt latency of one cycle.



case..

Figure 1. Interrupt Latency of One Cycle

In Figure 1 (a), the two single-cycle instructions are issued to their respective execution units and placed in the history buffer. During this same clock cycle, an external interrupt is received. Since there are no memory operations in progress, the interrupt is immediately detected and is subsequently recognized when the **add** instruction writes back. In this case, the interrupt is recognized one clock cycle after being received by the MC88110. In Figure 1 (b), the register file set is restored.

Example 2: The second case, as shown in Figure 2, depicts another short latency example.

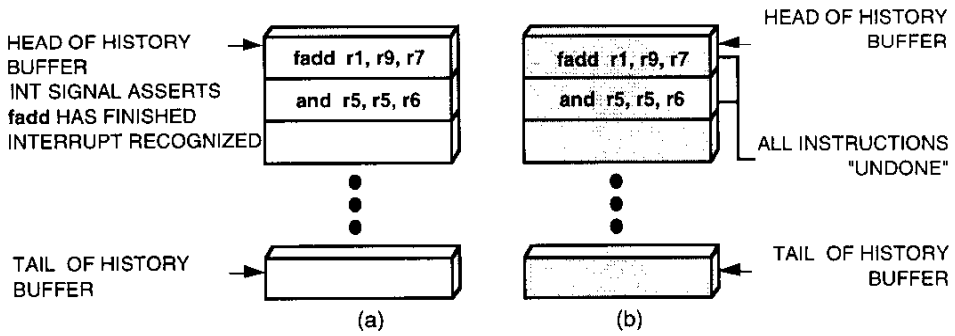


Figure 2. Short Latency Case

In Figure 2 (a), a three-cycle **fadd** instruction and a single-cycle instruction are issued to their respective execution units and placed in the history buffer. During this same clock cycle, an external interrupt is received. As in the previous example, since there are no memory operations in progress, the interrupt is immediately detected and is subsequently recognized three clock cycles later when the **fadd** instruction writes back. In this case, the interrupt is recognized three clock cycles after being received by the MC88110. In Figure 2 (b), the register file set is restored.

LONG INTERRUPT RECOGNITION LATENCY

There are several cases that result in long latencies for exception recognition. In this section, we present three examples of such cases. All three cases involve memory instructions (such as **ld**, **st**, or **xmem**) accessing either the cache (and has a cache miss) or the bus when an interrupt is detected. In the following paragraphs, we use the **ld** to illustrate these three long latency examples. In these examples, the **ld** instruction could easily have been a **st** or an **xmem**. Typically the **xmem** instruction provides the longest latency since it consists of both a load and a store. In the following examples described, the term **mw** is used in the latency expressions and represents the number of wait states of the computer's memory system. It is assumed that the MC88110 is parked and does not have to arbitrate for any bus.

Example 1: The first case (see Figure 3) involves a load instruction followed by a multi-cycle instruction that uses the destination register of the load instruction. We illustrate this by using the longest latency multi-cycle instruction—the floating-point divide (**fdiv**). If the load instruction is at the top of the buffer, and has already accessed the bus or cache when the interrupt is received, the load is allowed to complete before the process on interrupt recognition begins (illustrated in Figure 3 (a)). The MC88110 will initiate the interrupt recognition process one cycle after the load writes back. Since the **fdiv** has a dependency on the load, it will not issue until the load has completed execution. Once this data is available, the load writes back to the register file and is retired out of the history buffer. During the same clock cycle, the data is forwarded to the **fdiv** operation which is inserted into the history buffer (illustrated in Figure 3 (b)). One cycle after the load writes back, instruction issue is stalled. Note however that the **fdiv** has already been inserted into the history buffer and begun execution. When the **fdiv** attempts to write-back, the processor recognizes the interrupt, restores the processor state to the state before the **fdiv** was issued, and then fetches instructions from the interrupt handler.

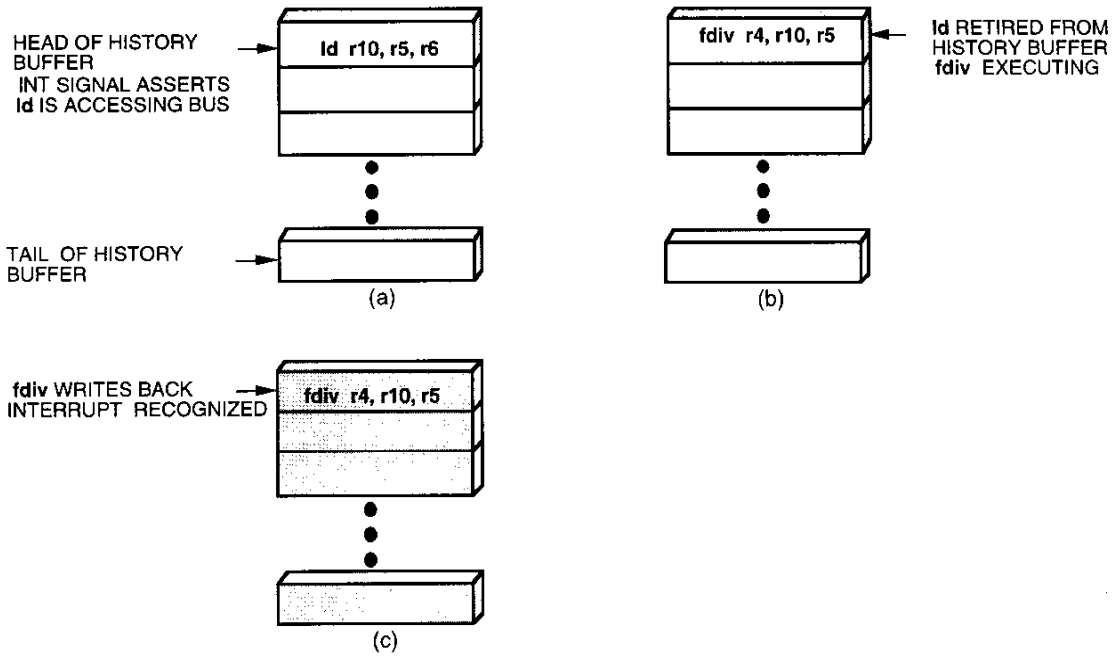
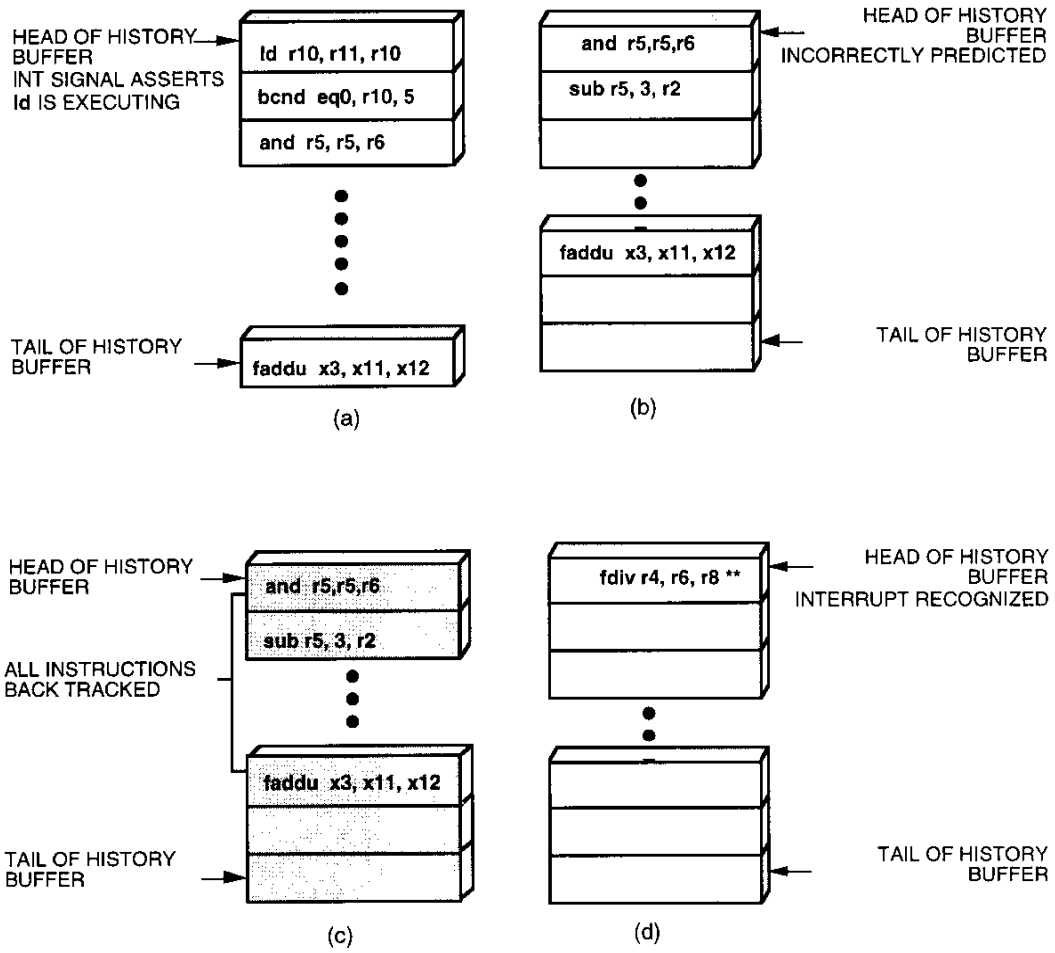


Figure 3. ld Followed by fdiv Instruction

In the example depicted in Figure 3, it takes **5 + mw** clocks for the load to complete and feed-forward the data to the divide unit. The **ld** takes **3 + mw** cycles to complete after it touches the cache. The floating-point divide instruction takes 23 cycles to complete. Thus, if the interrupt is detected by the MC88110 when the **ld** is accessing the cache, the interrupt would be recognized **[3 + mw] + 23** cycles after it is detected.

Example 2: The second long latency case (shown in Figure 4) involves a load that is immediately followed by an incorrectly predicted branch and ten instructions issued conditionally. A long latency occurs if the interrupt is detected while the load is accessing the bus. This situation is illustrated in Figure 4 (a). The process of interrupt recognition is initiated when the load operation completes and both the load operation and the branch are retired from the history buffer (as shown in Figure 4 (b)). Since the branch prediction was incorrect, the processor first has to back out of the instructions that were speculatively executed before the interrupt can be recognized. This is illustrated in Figure 4 (c). The MC88110 reverses program state at a rate of two instructions per clock cycle, resulting in five clock cycles of "back out" time. To recognize the interrupt, the first instruction that is fetched from the correct target of the branch (Figure 4 part (d)) is tagged as causing an unimplemented opcode exception. The MC88110 notices the exception tag and then recognizes the external interrupt. Note that the unsigned floating-point add (**fadd**) is the last instruction in Figure 4 (a), (b), and (c). Once the interrupt is detected and a load is already touching the bus, the processor will normally continue issuing instructions until the load completes its write-back. However, in this example, instruction issue stalls immediately because the history buffer is full.

As in the previous example, the latency of the load instruction will be **3 + mw**. Five clocks are required to back out the 10 incorrectly issued instructions following the branch. **5 + mw** cycles are required to fetch the instruction from the correct branch target (assuming a cache miss). This instruction is marked as causing the exception and the interrupt is recognized. So total latency for recognition is **[3 + mw] + 5 + [5 + mw]** cycles.



**This instruction is tagged with an unimplemented opcode exception.

Figure 4. Incorrectly Predicted Branch

Example 3: The third long latency scenario involves a **ld** that has accessed the bus, followed by instructions that fill up the two write-back slots of the MC88110. Since **ld** instructions have lowest priority for the write-back slots, the other instructions are allowed to write back before the load. In this case, interrupt recognition is delayed because the MC88110 will not initiate the process of interrupt recognition until the **ld** writes back. A long latency will occur if the history buffer contains a **ld** followed by a multi-cycle instruction that writes back the cycle before the load wants to perform its write-back. The multi-cycle instruction in turn is followed by several single-cycle instructions that write back, so that they prevent the **ld** from writing back. Under these circumstances, the interrupt cannot be recognized until all those instructions finish executing. Such a scenario is depicted in Figure 5. In this example, the **ld** is followed by an **fdiv** instruction, followed by eight single-cycle **add** instructions, the first of which has a dependency on the **fdiv**.

Figure 5(a)–(e) shows the contents of the history buffer. Instructions that have completed execution and are writing or have written back to the register file are shaded in the diagram. In Figure 5(a), the **ld** and the **fdiv** are the only executing instructions in the history buffer. The interrupt is signaled and detected by the MC88110 while the **ld** is accessing the bus. Instruction issue does not stop, however, until the **ld** completes and writes back to the register file. The first **add** instruction uses the destination register of the **fdiv** instruction, and thus cannot be issued until the **fdiv** finishes and is writing back. In Figure 5(b), the **fdiv** has finished execution and is writing back to the register file. In this cycle, the data for register **r4** is feed-forwarded to the **add** which then issues along with a second **add**. The **ld** also finishes execution and

prepares to write-back in the next cycle. Note that the **fdiv** cannot be retired from the history buffer until the **ld** has written back. In the next cycle shown in Figure 5(c), the **add** instructions that have finished execution write back their results to the register file. The **ld** cannot write-back because the two **add** instructions have higher write-back priority and use up the available write-back slots in the MC88110.

During the cycle in which the **ld** write-back is stalled, two additional **add** instructions are issued. These new **add** instructions write back the following cycle, again preventing the **ld** from writing back. More single-cycle **add** instructions are issued, further delaying the **ld** from writing back. This process continues until the history buffer is full, as shown in Figure 5(d). The last two instructions in the buffer are an **add** and an **fdiv**. Since the **fdiv** takes 23 cycles to execute, it will not write back at the same time as the **add** instruction, so the **ld** can utilize the free write-back slot and write back at the same time as the last **add** instruction. After writing back, the **ld** is retired from the history buffer along with the first **fdiv** and the nine consecutive **add** instructions, see Figure 5(e). Instruction issue is halted and the interrupt is recognized when the last **fdiv** writes back.

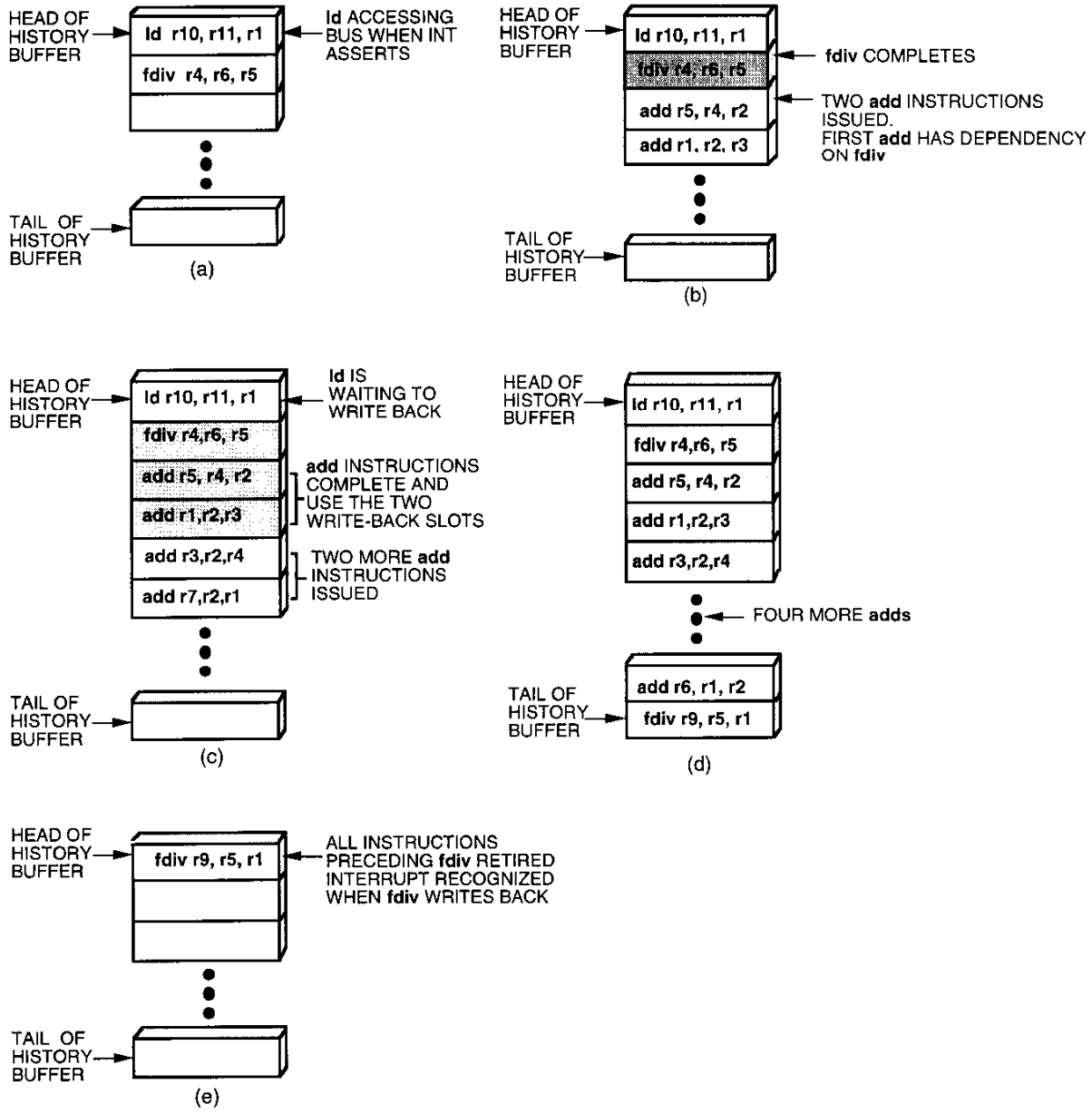


Figure 5. ld Waiting on Write-Back

The recognition latency for this case is longest if the **ld** finishes one cycle after the first **fdiv**. This will be the case if the **ld** takes 24 cycles to complete. Thus the recognition latency will include the time to complete the **ld** after it accesses the cache (22 cycles), the time delay before the **ld** can write back (4 cycles), and the time to complete the **fdiv** (22 cycles). Note that the **fdiv** is 22, not 23, cycles because one cycle of the **fdiv**'s execution overlaps with the four cycles of **ld** write-back delay. Therefore, the total latency in this case is **22 + 4 + 23 = 49** cycles.