Pipeline Stage	External Interrupts	Software Interrupts	Internal Interrupts
IF			
IS			
RF	Bus error	Breakpoint Syscall ECC instruction Virtual coherency instruction	Illegal instruction Instruction translation Coprocessor is unusable
EX	Interrupt		
DF			
DS		Floating point	Overflow
TC			TLB modified Data translation
WB	Data Watch NMI Reset	Virtual coherency	Bus error data

TABLE 9.2 R4000 stage designation of interrupted instruction.

stages are discussed in Chapter 10. However, note that a bus error on an instruction fetch from the cache is not signaled until the RF stage. The reason for this is that the data cache is pipelined and the checks of the cache tags occur in the RF stage as shown in Figure 2.31. For concurrent interrupts, the R4000 gives priority to the interrupted instruction furthest down the pipeline. This processor illustrates the need to recognize interrupts and save the program counter value over most of the stages of the pipeline.

9.1.2 *Handling Preceding Instructions*

The methods for handling preceding instructions, described below, are for systems that issue one instruction at a time. Systems that issue more than one instruction at a time are described in Chapter 10. Preceding instructions have the potential to modify the processor state in registers and/or memory. Recall that Smith's Condition #1 states that: All instructions preceding the instruction indicated by the saved program counter have been executed and have modified the process (processor) state correctly. This *sequential execution model* condition is defined as:

A processor that satisfies the condition that "the result of an execution is the same as if the operations had been executed in the order specified by the program" [LAMP79]. Note that a processor that executes the serial execution model also executes the sequential execution model.

The sequential execution model is enforced by one of the three design options for handling preceding instructions shown in Table 9.1. The options are:

1. to flush the pipeline, retiring all issued instructions;
2. to take steps to ensure that all issued instructions retire in-order;
3. to undo the processor state changes of any instructions that have been retired out-of-order.

The design issues involved in selecting the option for handling the preceding instructions are whether or not to (1) penalize the normal performance of the pipeline, as measured by CPI, at the expense of a longer interrupt latency when there is an interrupt, or (2) have a smaller CPI and a longer latency. Cost and complexity are also design considerations.

Only a Type A instruction window (described in Chapter 8) with in-order release will always have in-order retirements. The other types of windows have the potential for out-of-order completions and out-of-order retirements. Figure 9.3 shows windows with two execution units that can produce out-of-order retirements. Multiple execution units can have a different number of stages, such as an integer unit and a floating point unit as found in many microprocessors. Multiple register files, for integer and floating points, are not shown but have the same out-of-order retirement problem. Systems with a common result bus that store integer and floating point values in the same register file can have structural hazards on the output bus and the register file.

The operation of these windows and pipelines is illustrated with two examples of a four-instruction sequence using the long and the short execution units. For the Type C window, if an interrupt occurs at the end of t_4, i2 has already been retired out-of-order by writing to the register file. In addition, even if the interrupt had not occurred, there will be a structural hazard on the result bus and register file at t_5 that must be resolved.

For the Type D or E window, the instructions issue in-order to the reservation stations. For this example, instruction i3 is delayed two clocks because of a dependency on i2 (without forwarding). If an interrupt occurs at t_5, i2 will have been retired out-of-order. Notice that the release is out-of-order; i4 releases before i3 due to the dependency. The characteristics of these two configurations follow.

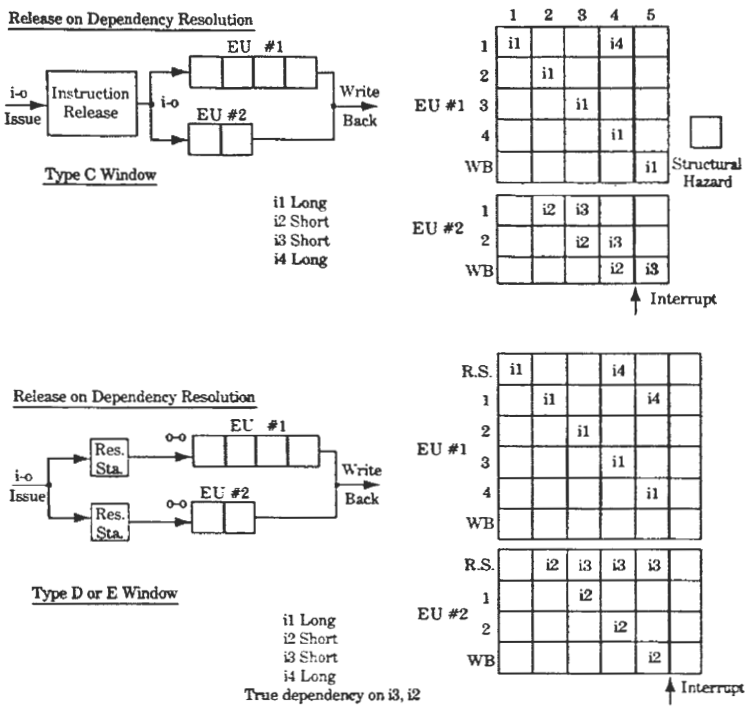


FIGURE 9.3 Multiple execution units.

Type C window (Cray-1, MIPS, Pentium integer) characteristics include

1. in-order issue,
2. dependencies resolved before release,
3. in-order release,
4. deterministic time from issue to completion.

Type D or E window (CDC 6600, IBM S/360/91 Flt. Pt., MC68110) characteristics include

1. in-order issue,
2. dependencies resolved in the reservation stations,
3. out-of-order release,
4. nondeterministic time from issue to completion.

Resolving Structural Hazards

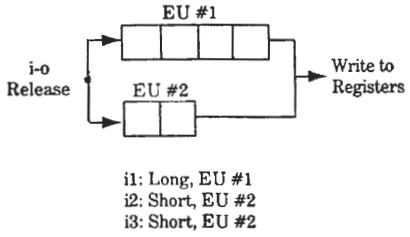
For systems with multiple-variable length execution units, the potential exists for a structural hazard on the result bus and register file even if instructions are released in-order. There are two methods for resolving this structural hazard.

1. Deterministically schedule the issue or release of instruction to eliminate the hazard.
2. Resolve the hazard with priority logic.

Consider the first method. Structural hazards can be resolved by delaying an instruction release until the hazard is eliminated; the result bus is scheduled so that hazards will not occur. Figure 9.4 shows a result shift register [SMITH85], called a RSR(a), that schedules the result bus and writes to a common register file.

The reservation table of a shift register, shown on the right of the figure, is the same length as the longest execution unit pipeline; in this case four stages, and the stages are numbered in reverse order. This result shift register reserves a time slot for the result bus and the path to the register file but does not ensure in-order retirement. Scheduling of instructions into the execution units is illustrated by a three-instruction sequence. Instruction i1, because it uses the long pipeline, places its destination address in stage four at t_1. Then i2 releases and places its destination address into stage 2 at t_2. Instruction i3 cannot release into EU #2 at t_3 because the bus will be blocked by the result of i1. Thus, i3 is delayed one clock and releases at t_4. When a result is ready to exit the execution unit, the result bus is gated on.

The TI ASC uses a Type C window and a RSR(a) to schedule its four execution units into the result bus and register file. Because of the number of data types, the RSR required seven bits to indicate the address and type of register. For example, an operation with an upper half-



	1	2	3	4	5	6	7	8
4	i1							
3		i1						
2		i2	i1	i3				
1			i2	i1	i3			
WB				i2	i1	i3		

* Reserves Bus and Register
* Does Not Preserve Order
* Requires 6 Clocks

FIGURE 9.4 Result shift register (a).

word result will not have a structural hazard with a result with a lower half-word result.

The second method for resolving the structural hazard employs priority logic. Type C, D, and E windows cannot easily use a RSR(a) to schedule the result bus because of the possible release delays while waiting for dependencies to be resolved. The CDC 6600 scoreboard and CDB must resolve these structural hazards on their result bus(ses) and move the delays to the completion end of the execution units. In the case of the scoreboard, each of the busses, called *trunks*, has priority hardware to delay lower priority bus requests. I am not certain, but I believe that some form of priority scheme is used with the CDB as well. The two floating point execution units probably receive CDB access priority based on a random selection.

Retirement by Flushing

After ensuring that there are no structural hazards, the requirements for properly handling the preceding instructions must be satisfied. A common method, used in a number of processors, is a simple flush of the pipelines; that is, the issued instructions in the pipelines are run to completion and all results are retired. After flushing, the processor state is correct, the context switch can be performed, and the interrupt serviced.

Flushing is illustrated with the reservation table in Figure 9.5. which shows an instruction sequence: long, short, long, short. The window can be Type C, D, or E.

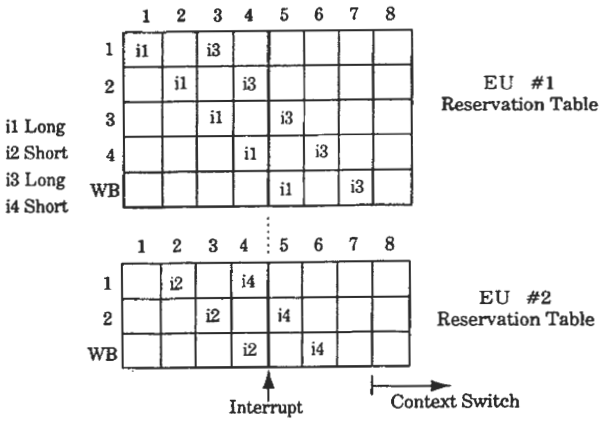


FIGURE 9.5 Flush pipelines.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.