# The CPU Pipeline

# 3

This chapter describes the basic operation of the CPU pipeline, which includes descriptions of the delay instructions (instructions that follow a branch or load instruction in the pipeline), interruptions to the pipeline flow caused by interlocks and exceptions, and R4400 implementation of an uncached store buffer.

The FPU pipeline is described in Chapter 6.

## 3.1 CPU Pipeline Operation

The CPU has an eight-stage instruction pipeline; each stage takes one PCycle (one cycle of PClock, which runs at twice the frequency of MasterClock). Thus, the execution of each instruction takes at least eight PCycles (four MasterClock cycles). An instruction can take longer—for example, if the required data is not in the cache, the data must be retrieved from main memory.

Once the pipeline has been filled, eight instructions are executed simultaneously. Figure 3-1 shows the eight stages of the instruction pipeline; the next section describes the pipeline stages.

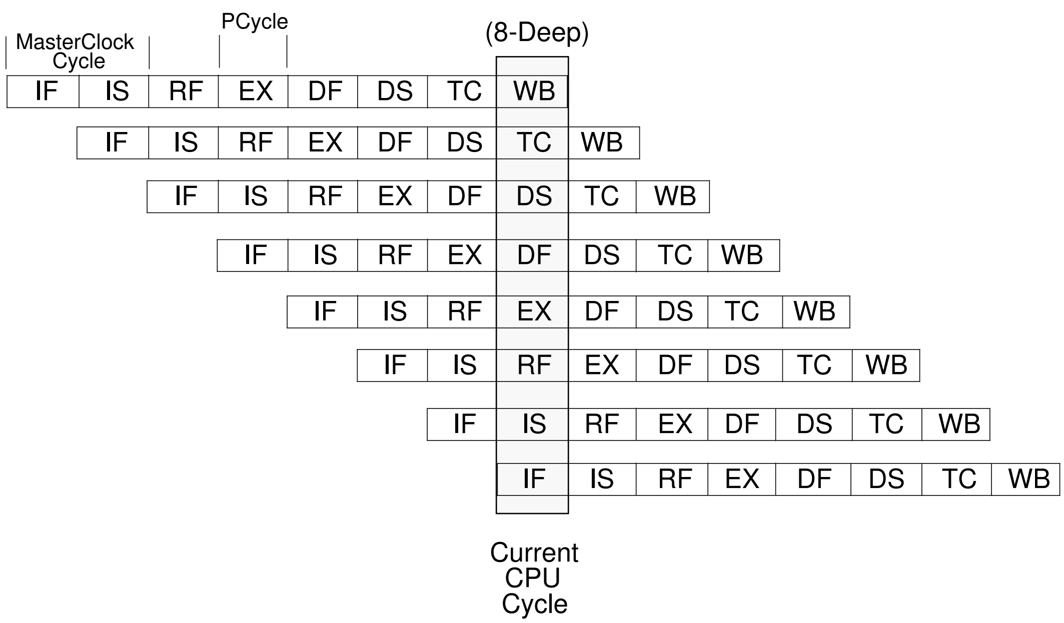



*Figure 3-1 Instruction Pipeline Stages*

## 3.2 CPU Pipeline Stages

This section describes each of the eight pipeline stages:

- IF - Instruction Fetch, First Half
- IS - Instruction Fetch, Second Half
- RF - Register Fetch
- EX - Execution
- DF - Data Fetch, First Half
- DS - Data Fetch, Second Half
- TC - Tag Check
- WB - Write Back

**IF - Instruction Fetch, First Half**

During the IF stage, the following occurs:

- Branch logic selects an instruction address and the instruction cache fetch begins.
- The instruction translation lookaside buffer (ITLB) begins the virtual-to-physical address translation.

**IS - Instruction Fetch, Second Half**

During the IS stage, the instruction cache fetch and the virtual-to-physical address translation are completed.

**RF - Register Fetch**

During the RF stage, the following occurs:

- The instruction decoder (IDEC) decodes the instruction and checks for interlock conditions.
- The instruction cache tag is checked against the page frame number obtained from the ITLB.
- Any required operands are fetched from the register file.

### EX - Execution

During the EX stage, one of the following occurs:

- The arithmetic logic unit (ALU) performs the arithmetic or logical operation for register-to-register instructions.
- The ALU calculates the data virtual address for load and store instructions.
- The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.

### DF - Data Fetch, First Half

During the DF stage, one of the following occurs:

- The data cache fetch and the data virtual-to-physical translation begins for load and store instructions.
- The branch instruction address translation and translation lookaside buffer (TLB)† update begins for branch instructions.
- No operations are performed during the DF, DS, and TC stages for register-to-register instructions.

### DS - Data Fetch, Second Half

During the DS stage, one of the following occurs:

- The data cache fetch and data virtual-to-physical translation are completed for load and store instructions. The Shifter aligns data to its word or doubleword boundary.
- The branch instruction address translation and TLB update are completed for branch instructions.

### TC - Tag Check

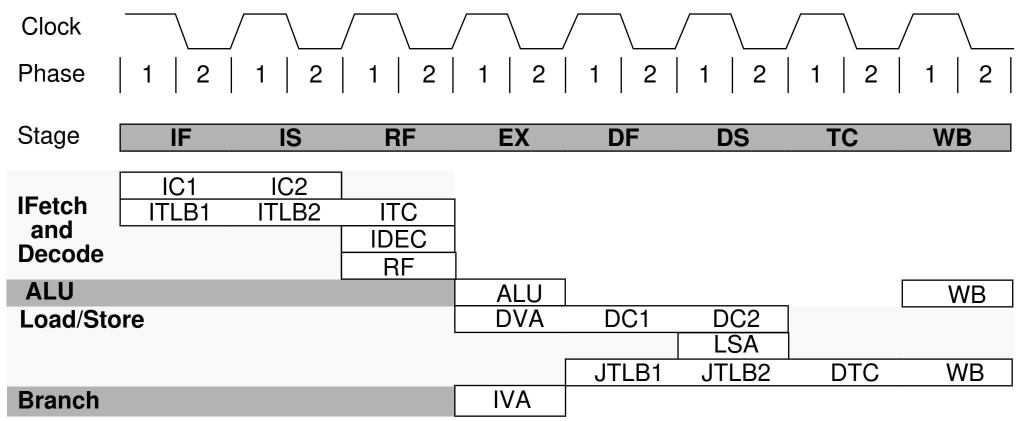
For load and store instructions, the cache performs the tag check during the TC stage. The physical address from the TLB is checked against the cache tag to determine if there is a hit or a miss.

† The TLB is described in Chapter 4.

### WB - Write Back

For register-to-register instructions, the instruction result is written back to the register file during the WB stage. Branch instructions perform no operation during this stage.

Figure 3-2 shows the activities occurring during each ALU pipeline stage, for load, store, and branch instructions.




| | |
|---|---|
| IC1 | Instruction cache access stage 1 |
| IC2 | Instruction cache access stage 2 |
| ITLB1 | Instruction address translation stage 1 |
| ITLB2 | Instruction address translation stage 2 |
| ITC | Instruction tag check |
| IDEC | Instruction decode |
| RF | Register operand fetch |
| ALU | Operation |
| DVA | Data virtual address calculation |
| DC1 | Data cache access stage 1 |
| DC2 | Data cache access stage 2 |
| LSA | Data load or store align |
| JTLB1 | Data/Instruction address translation stage 1 |
| JTLB2 | Data/Instruction address translation stage 2 |
| DTC | Data tag check |
| IVA | Instruction virtual address calculation |
| WB | Write back to register file |

*Figure 3-2 CPU Pipeline Activities*

## 3.3 Branch Delay

The CPU pipeline has a branch delay of three cycles and a load delay of two cycles. The three-cycle branch delay is a result of the branch comparison logic operating during the EX pipeline stage of the branch, producing an instruction address that is available in the IF stage, four instructions later.

Figure 3-3 illustrates the branch delay.

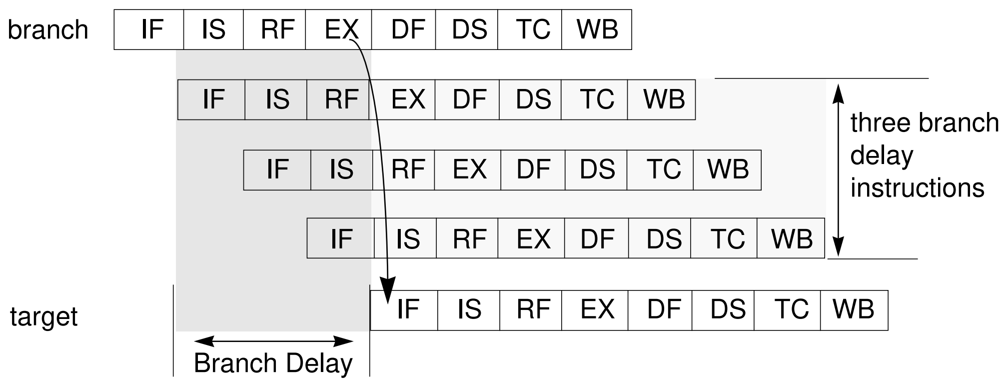



*Figure 3-3 CPU Pipeline Branch Delay*

## 3.4 Load Delay

The completion of a load at the end of the DS pipeline stage produces an operand that is available for the EX pipeline stage of the third subsequent instruction.

Figure 3-4 shows the load delay of two pipeline stages.

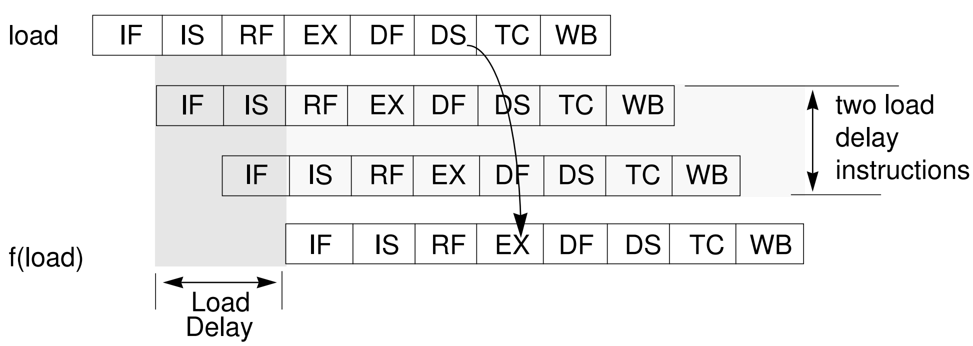



*Figure 3-4 CPU Pipeline Load Delay*

## 3.5 Interlock and Exception Handling

Smooth pipeline flow is interrupted when cache misses or exceptions occur, or when data dependencies are detected. Interruptions handled using hardware, such as cache misses, are referred to as *interlocks*, while those that are handled using software are called exceptions.

As shown in Figure 3-5, all interlock and exception conditions are collectively referred to as faults.

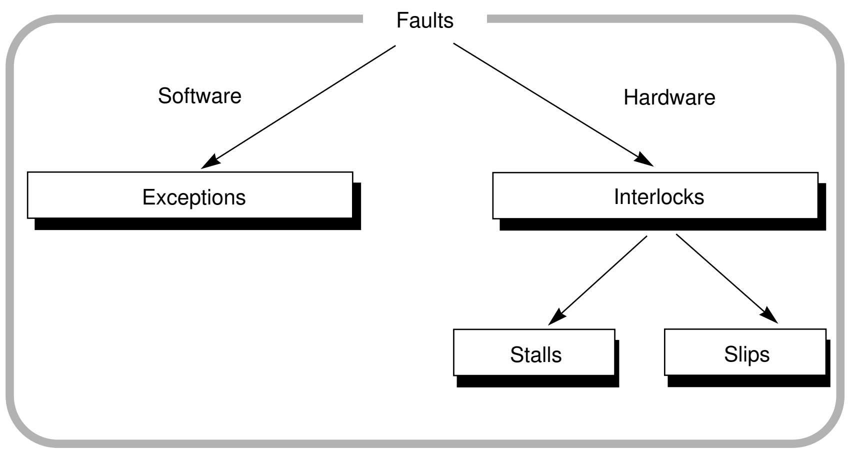



*Figure 3-5 Interlocks, Exceptions, and Faults*

There are two types of interlocks:

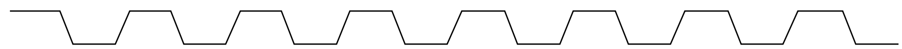
- stalls, which are resolved by halting the pipeline
- slips, which require one part of the pipeline to advance while another part of the pipeline is held static

At each cycle, exception and interlock conditions are checked for all active instructions.

Because each exception or interlock condition corresponds to a particular pipeline stage, a condition can be traced back to the particular instruction in the exception/interlock stage, as shown in Figure 3-6. For instance, an Illegal Instruction (II) exception is raised in the execution (EX) stage.

Tables 3-1 and 3-2 describe the pipeline interlocks and exceptions listed in Figure 3-6.

Clock



PCycle | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 | 1 | 2 |

| **State** | **Pipeline Stage** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **IF** | **IS** | **RF** | **EX** | **DF** | **DS** | **TC** | **WB** |
| Stall* | | ITM | ICM | | CPBE | | DCM | |
| | | | | | SXT | | WA | |
| | | | | | STI | | | |
| *MP stalls can occur at any stage; they are not associated with any instruction or pipe stage | | | | | | | | |
| | **IF** | **IS** | **RF** | **EX** | **DF** | **DS** | **TC** | **WB** |
| Slip | | | LDI | | | | | |
| | | | MultB | | | | | |
| | | | DivB | | | | | |
| | | | MDOne | | | | | |
| | | | ShSlip | | | | | |
| | | | FCBsy | | | | | |
| | **IF** | **IS** | **RF** | **EX** | **DF** | **DS** | **TC** | **WB** |
| Exceptions | | | ITLB | Intr | | OVF | DTLB | DBE |
| | | | | IBE | | FPE | TLBMod | Watch |
| | | | | IVACoh | | ExTrap | | DVACoh |
| | | | | II | | | | DECCErr |
| | | | | BP | | | | NMI |
| | | | | SC | | | | Reset |
| | | | | CUn | | | | |
| | | | | IECCErr | | | | |

*Figure 3-6 Correspondence of Pipeline Stage to Interlock Condition*

*Table 3-1 Pipeline Exceptions*

| Exception | Description |
|---|---|
| ITLB | Instruction Translation or Address Exception |
| Intr | External Interrupt |
| IBE | IBus Error |
| IVACoh | IVA Coherent |
| II | Illegal Instruction |
| BP | Breakpoint |
| SC | System Call |
| CUn | Coprocessor Unusable |
| IECCErr | Instruction ECC Error |
| OVF | Integer Overflow |
| FPE | FP Interrupt |
| ExTrap | EX Stage Traps |
| DTLB | Data Translation or Address Exception |
| TLBMod | TLB Modified |
| DBE | Data Bus Error |
| Watch | Memory Reference Address Compare |
| DVACoh | DVA Coherent |
| DECCErr | Data ECC Error |
| NMI | Non-maskable Interrupt |
| Reset | Reset |

*Table 3-2 Pipeline Interlocks*

| Interlock | Description |
|---|---|
| ITM | Instruction TLB Miss |
| ICM | Instruction Cache Miss |
| CPBE | Coprocessor Possible Exception |
| SXT | Integer Sign Extend |
| STI | Store Interlock |
| DCM | Data Cache Miss |
| WA | Watch Address Exception |
| LDI | Load Interlock |
| MultB | Multiply Unit Busy |
| DivB | Divide Unit Busy |
| MDOne | Mult/Div One Cycle Slip |
| ShSlip | Var Shift or Shift > 32 bits |
| FCBsy | FP Busy |

## Exception Conditions

When an exception condition occurs, the relevant instruction and all those that follow it in the pipeline are cancelled. Accordingly, any stall conditions and any later exception conditions that may have referenced this instruction are inhibited; there is no benefit in servicing stalls for a cancelled instruction.

After instruction cancellation, a new instruction stream begins, starting execution at a predefined exception vector. System Control Coprocessor registers are loaded with information that identifies the type of exception and auxiliary information such as the virtual address at which translation exceptions occur.

## Stall Conditions

Often, a stall condition is only detected after parts of the pipeline have advanced using incorrect data; this is called a *pipeline overrun*. When a stall condition is detected, all eight instructions—each different stage of the pipeline—are frozen at once. In this stalled state, no pipeline stages can advance until the interlock condition is resolved.

Once the interlock is removed, the restart sequence begins two cycles before the pipeline resumes execution. The restart sequence reverses the pipeline overrun by inserting the correct information into the pipeline.

## Slip Conditions

When a slip condition is detected, pipeline stages that must advance to resolve the dependency continue to be retired (completed), while dependent stages are held until the required data is available.

## External Stalls

*External stall* is another class of interlocks. An external stall originates outside the processor and is not referenced to a particular pipeline stage. This interlock is not affected by exceptions.

## Interlock and Exception Timing

To prevent interlock and exception handling from adversely affecting the processor cycle time, the R4000 processor uses both logic and circuit pipeline techniques to reduce critical timing paths. Interlock and exception handling have the following effects on the pipeline:

- In some cases, the processor pipeline must be backed up (reversed and started over again from a prior stage) to recover from interlocks.
- In some cases, interlocks are serviced for instructions that will be aborted, due to an exception.

These two cases are discussed below.

## Backing Up the Pipeline

An example of pipeline back-up occurs in a data cache miss, in which the late detection of the miss causes a subsequent instruction to compute an incorrect result.

When this occurs, not only must the cache miss be serviced but the EX stage of the dependent instruction must be re-executed before the pipeline can be restarted. Figure 3-7 illustrates this procedure; a minus (–) after the pipeline stage descriptor (for instance, EX–) indicates the operation produced an incorrect result, while a plus (+) indicates the successful re-execution of that operation.

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cycle | Run | Run | Run | Run | Run | Run | Run | Stl | Stl | Stl | Stl | Stl | Run | Run | Run | Run | Run |
| Restart | | | | | | | | | | | Rst2 | Rst1 | | | | | |
| Load | IF | IS | RF | EX | DF | DS | TC | | | DF | DS | TC | WB | | | | |
| | | IF | IS | RF | EX | DF | DS | | | | DF | DS | TC | WB | | | |
| ALU | | | IF | IS | RF | EX | DF | | | | | DF | DS | TC | WB | | |
| | | | | IF | IS | RF | EX- | | | | RF | EX+ | DF | DS | TC | WB | |
| | | | | | IF | IS | RF | | | | | | EX | DF | DS | TC | WB |

*Figure 3-7 Pipeline Overrun*

### Aborting an Instruction Subsequent to an Interlock

The interaction between an integer overflow and an instruction cache miss is an example of an interlock being serviced for an instruction that is subsequently aborted.

In this case, pipelining the overflow exception handling into the DF stage allows an instruction cache miss to occur on the next immediate instruction. Figure 3-8 illustrates this; aborted instructions are indicated with an asterisk (*).

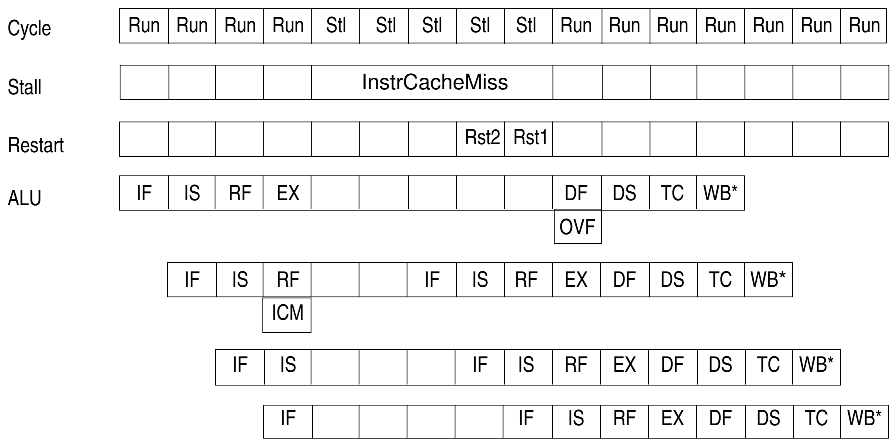



*Figure 3-8 Instruction Cache Miss*

Even though the line brought in by the instruction cache could have been replaced by a line of the exception handler, no performance loss occurs, since the instruction cache miss would have been serviced anyway, after returning from the exception handler. Handling of the exception is done in this fashion because the frequency of an exception occurring is, by definition, relatively low.

## Pipelining the Exception Handling

Pipelining of interlock and exception handling is done by pipelining the logical resolution of possible fault conditions with the buffering and distributing of the pipeline control signals.

In particular, a half clock period is provided for buffering and distributing the *run* control signal; during this time the logic evaluation to produce run for the next cycle begins. Figure 3-9 shows this process for a sequence of loads.

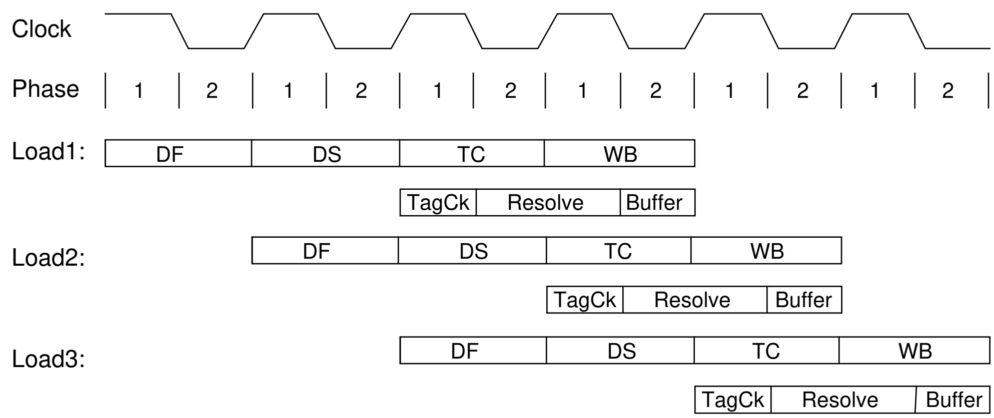



*Figure 3-9 Pipelining of Interlock and Exception Handling*

The decision whether or not to advance the pipeline is derived from these three rules:

- All possible fault-causing events, such as cache misses, translation exceptions, load interlocks, etc., must be individually evaluated.
- The fault to be serviced is selected, based on a predefined priority as determined by the pipeline stage of the asserted faults.
- Pipeline advance control signals are buffered and distributed.

Figure 3-10 illustrates this process.

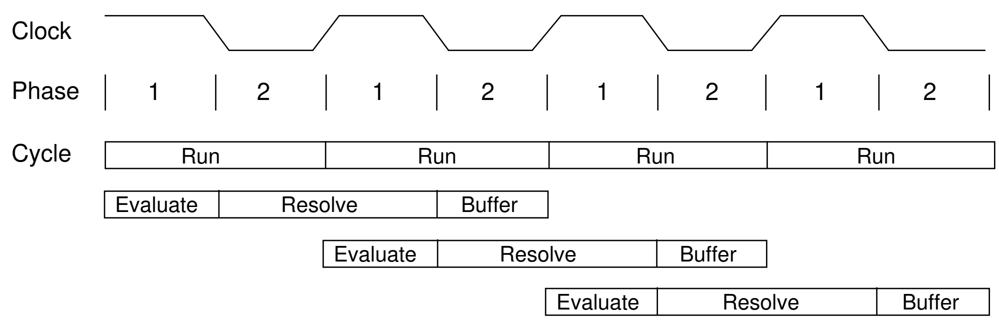



*Figure 3-10 Pipeline Advance Decision*

# Special Cases

In some instances, the pipeline control state machine is bypassed. This occurs due to performance considerations or to correctness considerations, which are described in the following sections.

## Performance Considerations

A performance consideration occurs when there is a cache load miss. By bypassing the pipeline state machine, it is possible to eliminate up to two cycles of load miss latency. Two techniques, *address acceleration* and *address prediction*, increase performance.

### Address Acceleration

*Address acceleration* bypasses a potential cache miss address. It is relatively straightforward to perform this bypass since sending the cache miss address to the secondary cache has no negative impact even if a subsequent exception nullifies the effect of this cache access. Power is wasted when the miss is inhibited by some fault, but this is a minor effect.

### Address Prediction

Another technique used to reduce miss latency is the automatic increment and transmission of instruction miss addresses following an instruction cache miss. This form of latency reduction is called *address prediction*: the subsequent instruction miss address is predicted to be a simple increment of the previous miss address. Figure 3-11 shows a cache miss in which the cache miss address is changed based on the detection of the miss.

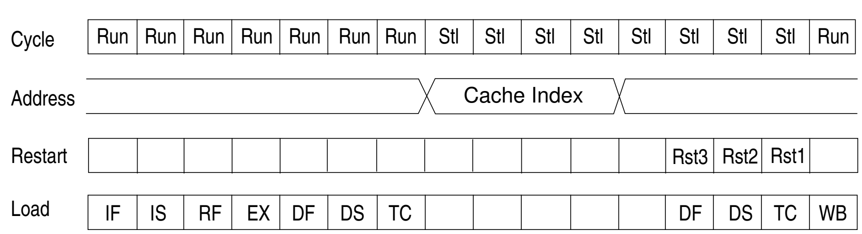



*Figure 3-11 Load Address Bypassing*

## Correctness Considerations

An example in which bypassing is necessary to guarantee correctness is a cache write.

## 3.6 R4400 Processor Uncached Store Buffer

The R4400 processor contains an uncached store buffer to improve the performance of uncached stores over that available from an R4000 processor. When an uncached store reaches the write-back (WB) stage in the CPU pipeline, the CPU must stall until the store is sent off-chip. In the R4400 processor, a single-entry buffer stores this uncached WB-stage data on the chip without stalling the pipeline.

If a second uncached store reaches the WB stage in the R4400 processor before the first uncached store has been moved off-chip, the CPU stalls until the store buffer completes the first uncached store. To avoid this stall, the compiler can insert seven instruction cycles between the two uncached stores, as shown in Figure 3-12. A single instruction that requires seven cycles to complete could be used in place of the seven No Operation (NOP) instructions.

```
SW R2, (r3)        # uncached store
NOP                # NOP 1
NOP                # NOP 2
NOP                # NOP 3
NOP                # NOP 4
NOP                # NOP 5
NOP                # NOP 6
NOP                # NOP 7
SW R2, (R3)        # uncached store
```

*Figure 3-12 Pipeline Sequence for Back-to-Back Uncached Stores*

If the two uncached stores execute within a loop, the two killed instructions which are part of the loop branch latency are included in the count of seven interpolated cycles. Figure 3-13 shows the four NOP instructions that need to be scheduled in this case.

```
Loop:   SW R2, (R3)          # uncached store
        NOP
        NOP
        NOP
        B Loop               # branch to loop
        NOP
        killed               # branch latency
        killed               # branch latency
```

*Figure 3-13 Back-to-Back Uncached Stores in a Loop*

The timing requirements of the System interface govern the latency between uncached stores; back-to-back stores can be sent across the interface at a maximum rate of one store for every four external cycles. If the R4400 processor is programmed to run in divide-by-2 mode (for more information about divided clock, see the description of **SClock** in Chapter 10), an uncached store can occur every eight pipeline cycles. If a larger clock divisor is used, more pipeline cycles are required for each store.

**CAUTION:** The R4000 processor always had a strongly-ordered execution; however, with the addition of the uncached store buffer in the R4400 there is a potential for out-of-order execution (described in the section of the same name in Chapter 11, and Uncached Loads or Stores in Chapter 12).