Coprocessor 0 Hazards

F

The contents of the System Coprocessor registers and the TLB affect the operation of the processor in many ways. For instance, an instruction that changes CP0 data also affects subsequent instructions that use the data.

In the CPU, general registers are interlocked and the result of an instruction can generally be used by the next instruction; if the result is not available right away, the processor stalls until it is available. CP0 registers and the TLB are not interlocked, however; there may be some delay before a value written by one instruction is available to following instructions.

There is a *required-data dependence* between an instruction that changes a register or TLB entry (a *writer*) and the next instruction that uses it (a *user*). (A writer can write multiple data items, forming multiple writer/user pairs.) The writer/user instruction pair places a *hazard* on the data if there must be a delay between the time the writer instruction writes the data, and the user instruction can use the data.

In addition to instructions, events can be writers and users of CP0 information. For instance, an exception writes information to CP0 registers and events that occur for every instruction, like an instruction fetch, use CP0 information. Therefore, when manipulating CP0 contents, the systems programmer must identify hazards and write code that avoids these hazards.

Table F-1 describes how to identify and avoid hazards, listing instructions and events that use CP0 registers and the TLB. This table also tells when written information is available (column 3) and when this latest information can actually be used (column 2). *Exception event writer timing* refers to the instruction identified with the exception; *user event timing* information is the pipestage of each instruction during which the user event uses the data. In the case of a hazard, the number of instructions required between a writer and user is:

available_stage - (use_stage + 1)

To identify a hazard, look for an instruction/event writer/user pair that has a required-data dependence and use the timing information in the table to calculate the delay required between the writer and user. If no delay is required, there is no hazard. If there is a hazard, place enough instructions between the writer and user so that the written information is available or effective when the user needs it.

> **NOTE:** Any instructions inserted between a writer/reader pair with a hazard must not depend on or modify the data creating the hazard (for example NOP instructions may be used).

The following steps are used to determine a hazard delay:

1. Find the pipeline stage of the *writer* instruction in which the result is available. For example, the MTC0 instruction writes a CP0 general register, and the new value is available at stage 7.
2. Find the pipeline stage in which the *user* instruction reads or uses the data item that the writer changes. The TLBWR instruction, for example, uses different registers through different stages; all source register values must be stable by stage 5 and remain unchanged through stage 8.
3. Calculate the number of instructions that must be inserted between the hazardous pair, by using this formula: *available_stage - (use_stage + 1)*. For example, with an MTC0/TLBWR pair, MTC0 data is available at stage 7, and TLBWR data must be stable by stage 5 so the computation is: 7 - (5 + 1) = 1. This means 1 instruction must be inserted between the MTC0 and TLBWR. If the result of the computation is less than or equal to zero, there is no hazard and no instructions are required between the pair.

Table F-1 R4000 Coprocessor 0 Data Writer and User Timing

Instruction or Event	CP0 Data Used, Stage Used		CP0 Data Written, Stage Available	
MTC0 / DMTC0			CPR[0,rd]	7γδ
MFC0 / DMFC0	CPR[0,rd]	4βγ		
TLBR	Index, TLB	5-7	PageMask, EntryHi, EntryLo0, EntryLo1	8
TLBWI TLBWR	Index or Random, PageMask, EntryHi, EntryLo0, EntryLo1	5-8	TLB	8
TLBP	PageMask, EntryHi	3-6	Index	7
ERET	EPC or ErrorEPC, TLB	4	Status[EXL, ERL]	4-8α
	Status	3	LLbit	7
Index Load Tag			TagLo, TagHi, ECC	8βε
Index Store Tag	TagLo, TagHi, ECC	8ε		
CACHE Hit ops			Status[CH]	8ε
CACHE ops	cache line (see note)	ε	cache line (see note)	ε
Load/Store	EntryHi.ASID Status[KSU, EXL, ERL, RE], Config[K0, DB], TLB	4		
	Config[SB]	7		
	WatchHi, WatchLo	4-5		
Load/Store exception			EPC, Status, Cause, BadVaddr, Context, XContext	8
Instruction fetch exception			EPC, Status	8
			Cause, BadVAddr, Context, XContext	4
Instruction fetch	EntryHi[ASID], Status[KSU, EXL, ERL, RE], Config[K0, IB]	0α		
	Config.SB	3		
	TLB (mapped addresses)	2		
Coproc. usable test	Status[CU, KSU, EXL, ERL]	2		
Interrupt signals sampled	Cause[IP], Status[IM, IE, EXL, ERL]	3		
TLB shutdown			Status.TS	7

EntryHi.ASID refers to the *ASID* field of the *EntryHi* register.
Config[K0, DB] refers to the *K0* and *DB* fields of the *Config* register.

α The *EXL* and *ERL* bits in the *Status* register are permanently cleared in stage 8, if no exceptions abort the ERET. However the effect of clearing them is visible to an instruction fetch starting in stage 4, so the "returned to" instructions use the modified values in the *Status* register.

β Only one instruction is needed to separate Index Load Tag and MFC0 Tag, even though table timing indicates otherwise.

γ An MTC0 of a CPR must not be immediately followed by MFC0 of the same CPR.

δ With an MTC0 to *Status* that modifies *KSU* and sets *EXL* or *ERL*, it is possible for the five instructions following the MTC0 to be executed incorrectly in the new mode, and not correctly in the kernel mode. This can be avoided by setting *EXL* first, and only later changing the value of *KSU*.

ε There must be two non-load, non-CACHE instructions between a store and a CACHE instruction directed to the same primary cache line as the store.

Table F-2 lists some hazard conditions, and the number of instructions that must come between the writer and the user. The table shows the data item that creates the hazard, and the calculation for the required number of intervening instructions.

Table F-2 CP0 Hazards and Calculated Delay Times.

Writer	**→**	**User**	**Hazard On**	**Instructions Between**	**Calculation**
TLBWR/ TLBWI	→	TLBP	TLB entry	3	8-(4+1)
TLBWR/ TLBWI	→	load/store using new TLB entry	TLB entry	3	8-(4+1)
TLBWR/ TLBWI	→	I-fetch using new TLB entry	TLB entry	5	8-(2+1)
MTCO Status[CU]	→	Coprocessor instruction needs CU set	Status[CU]	4	7-(2+1)
TLBR	→	MFC0 EntryHi	EntryHi	3	8-(4+1)
MTC0 EntryLo0	→	TLBWR/TLBWI	EntryLo0	1	7-(5+1)
TLBP	→	MFC0 Index	Index	2	7-(4+1)
MTC0 EntryHi	→	TLBP	EntryHi	1	7-(5+1)
MTC0 EPC	→	ERET	EPC	2	7-(4+1)
MTC0 Status	→	ERET	Status	3	7-(3+1)
MTC0 Status[IE]	→	instruction interrupted†	Status[IE]	3	7-(3+1)

†. You cannot depend on a delay in effect if the instruction execution order is changed by exceptions. In this case, for example, the minimum delay for IE to be effective is the *maximum* delay before a pending, enabled interrupt can occur.