MIPS R4400MC Errata, Processor Revision 1.0

May 10, 1994

MIPS Technologies Inc.
2011 N Shoreline Blvd
PO Box 7311
Mountain View, CA 94039-7311

Additional errata which affect uniprocessor designs and may affect multiprocessor configurations are listed in the MIPS R4400 PC, R4400 SC errata. Change bars in the left column indicate corrections or changes from the last version of the errata.

1. The external interface specification indicates that it is possible to split the command and data cycles of an external update command. However, the R4400 will prematurely take ownership of the system interface bus before the data cycle is issued to the R4400 if there are any cycles between the update command cycle and the update data cycle.

Workaround: Do not split the command and data cycles of an external update command.

2. Under the conditions listed below, the EB bit in the CacheErr register is incorrectly set.

1) A store targets a shared line in the primary cache
2) The tag in this line has a parity error
3) Under this condition, the processor will stall due to a data cache miss and the CacheErr register is set.
4) As the processor comes out of the data cache miss and before it vectors to the CacheErr exception vector, there is an instruction cache miss and a pending external request which targets the same line with the parity error.

Under these conditions, the EB bit will get set although there was no parity error. The EB bit implies that both a data and instruction parity error have occurred. In this case, there was only a data parity error.

Workaround: The EB bit is meaningful only if the ER bit in the CacheErr register indicates an instruction error.

3. Processors might not function in "lock-step" properly because the timer in CP0 (Count Register) may not synchronize across multiple processors at reset. As a result, the timers may increment on different clock edges causing the processors to fall out of lock step.

Workaround: Do not use the Count Register as a timer if more than one processor needs to function in lock-step with each other.

4. Under the following conditions, a cache operation on primary cache may corrupt data in the secondary cache.

1. A primary cache cache operations stalls because a write back is required.
2. An Intervention request" is accepted by the processor to invalidate the same target line

The processor issues the correct data in response to the intervention request. But after the response is completed, the processor tries to complete the writeback of the invalidated data and will, incorrectly, set the SCAddr to 0 corrupting that secondary cache line.

Workaround: Do not use cache operations which involve a writeback on primary cache (Hit/Index_Writeback_Inv_D or Hit/Index_Writeback_D) , except when the conditions listed above cannot occur. These cache operations can be synthesized by executing a dummy load (lw r0, (rx)) to an address, which would map to the same primary line but to different secondary line. This load would then writeback the dirty primary data to the correct secondary line.

Another solution is to use the corresponding cache-operations on the secondary cache instead of the primary cache, which would ensure correctness, at the expense of some performance hit.

5. Under following condition, the DADDIU instruction can produce an incorrect result.
If this instruction generates a result value that would cause an overflow condition to occur (even though this instruction does not take an overflow exception) then the result value will be correct in bits 0-31 but bit 31 will be replicated through bits 32-63 (so it looks like a 32bit sign-extended value). The overflow condition is defined when the carries out of bits 62 and 63 differ (two's compliment overflow).

Workaround: There is no workaround for this problem.

6. Dirty shared mode may generate incorrect command sequences.
The problem occurs when the following sequence of events takes place:

1) store issued to a Dirty Shared or Shared line (line A)
2) processor invalidate is initiated and held up because of the deassertion of RdRdy*
3) an external request to the same or a different line is received (lineB).
4) RdRdy* is asserted.

At this point the processor should reissue the invalidate to line; however, it, incorrectly, issues one of the two requests:
1) in the Shared case, it issues coherent read to lineA
2) in the Dirty Shared case, it issues read with write forthcoming to lineA

The first case causes extra traffic but no serious problem; however, the second case, could be fatal since the processor issues the read for a line which it has modified and owns. The processor could end up with wrong data unless the read response uses the write data supplied by the processor.

Workaround: Do not use Dirty-Shared mode if RdRdy is used to control processor requests.

7. When a TLB refill exception occurs on an instruction fetch, the value in the CP0 register BadVAddr might not match CP0 register EPC (or EPC+4 in case of a branch or jump with the delay slot as the first instruction of the next page.

Workaround:

In the first level tlb refill exception, use the tlb probe instruction to check if the virtual address in the BadVAddr register already exist in the TLB. If it is in the TLB, then eret (as the BadVAddr was incorrect), else go ahead and write the new TLB entry and eret. By overlapping the TLB probe operation with the other instructions in the handler, and then placing the TLB write instruction in the branch delay slot of a branch likely instruction, the performance overhead for this workaround can be minimized.

Example:

```
        mfc0    k0,   context
        lw      k1,   0(k0)
        lw      k0,   4(k0)
        c0      tlbp            <-- additional instruction due to workaround
        srl     k1,   k1, 3
        mtc0    k1,   tlblo0
        srl     k0,   k0, 3
        mfc0    k1,   index     <-- additional instruction due to workaround
```

```
        mtc0    k0,  tlblo1
        bltzl   k1,  1f          <-- additional instruction due to workaround
        c0      tlbwr
1:      nop
        c0      eret
```

If the processor takes a TLB refill exeception from the first level exception then it will jump to the "general exception handler". Inside the "general exception handler", when the operating system (OS) detects an address outside the expected range in BadVAddr, it should check EPC to make sure it is within a valid range for the process. If EPC is within the valid range, the OS should execute an "eret" instruction. The refill instruction will be re-taken and BadVAddr will contain the correct value.

If the OS is unable to determine the valid address range for the process, the value in EPC should be used to look for a load or store instruction. If EPC does not point to a load or store, the OS should execute an "eret". The "eret" will then cause another TLB refill exception, which will have a valid BadVAddr. If EPC points to a load or store, the OS must then interpret the instruction to generate the address for the data. If this address matches the address in BadVAddr, the process tried to access data outside the process address space. Otherwise the OS should execute an "eret" causing a TLB refill exception where the value in BadVAddr will be valid.

8. When Create-Dirty-Exclusive-SD cacheop is performed on a line which is present in the processor in Shared or Dirty Shared state; the processor invalidates the line before modifying it to Dirty Exclusive state. This might create the following problem: If there is a snoop or an intervention to this line, while the processor is waiting for IvdAck, the processor could send an incorrect response with an Invalid state instead of Shared or DShared state.

Workaround:There is no workaround for this problem.

9. External Updates to a line, which exists both in the PICache and PDCache at the same time, causes the copy of the line in the SCache and PDCache to be updated but does not change the state of the copy in the PICache. If the line is in the SCache and the PICache, only, then the processor properly updates the SCache line and invalidates the PICache line; and if the line is in the SCache and the PDCache, only, then the line gets updated in both secondary and primary caches, as expected.

Workaround: Do not allow a line to exist in both PDCache and PICache if an update protocol is used or use “write invalidate” protocol for the instruction space.

10. In this following sequence:

```
ddiv        (or ddivu or div or divu)
dsll32      (or dsrl32, dsra32)
```

if an MPT stall occurs, while the divide is slipping the cpu pipeline, then the following double shift would end up with an incorrect result.

Workaround: The compiler needs to avoid generating any sequence with divide followed by extended double shift.

11. The processor sends Read Request with incorrect value of the *“Link Address Retained”* bit. This error occurs when the following sequence of events takes place:

1) ICache miss to a line replacing link address in scache.
2) ICache Read request is stopped by de-asserting RdRdyB
3) An external request comes during this time and R4400 has to regenerate the address and command.

When the processor regenerates the Read Request after responding to the external request, it compares the link address register with a different address than the instruction address that caused the miss. As a result, sometimes it incorrectly sets or resets the *"Link Address Retained"* bit.

The consequence of incorrectly setting the *"Link Address Retained"* bit are not of any concern since the external agent would snoop assuming the line exist in shared state; but the processor would provide the state as Invalid. However, the consequence of incorrectly not indicating the *"Link Address Retained"* is significant since the atomic functionality could be broken.

Workaround: The hardware solution is to either not use the RdRdyB signal or in the case when the RdRdyB is used, to latch the retained bit when it occurs with the first Read Request even though the request is not accepted.