* Exception Scheme in QS-I CPU

QS-I CPU has implemented some registers used in the handling of exception defined in MIPS architecture, though with some slight changes. Users can use instructions MFC0 and MTC0 to read or write these registers. Such as,

MFC0 $t0, Cause

MTC0 $t1, Cause

Status (#12):

31 30 29 28 27 9 8 4 3 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IE |  | Interrupt Mask | Reserved | x86 Flags |

Cause (#13):

31 30 29 28 27 9 8 7 6 2 1 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BR |  | Interrupt Pending |  | EXC |  |

IE: Interrupt Enable. If this bit is clear, then all interrupts will be disabled.

Interrupt Mask: this part is 20 bits wide, respectively indicates whether the up to 20 external interrupts should be masked or not. This part will be “and” with the interrupt pending part of the Cause register, and the priority of the 20 external interrupts is handled by software.

BR: Branch. If the instruction executing in the pipeline is a branch, then the BR will be asserted when any exception happens.

Interrupt Pending: this field indicates the pending state of the up to 20 external interrupts. Note that whenever an interrupt is serviced by software, the pending bit of the interrupt pending filed should be cleared.

EXC: Exception Code. When any exception happens, the exception code will be written into the Cause(#13) system register. And software can use the information provided by Cause register to determine which ISR to invoke.

EPC(#14):

|  |
| --- |
| Exception PC |

When any exception happens, the PC of the victim instruction will be written into the EPC register. And when leaving from the ISR, software needs to read from EPC and then goes back to the interrupted instruction.

**What exactly should software do whenever an exception occurs? The steps should be taken are listed below:**

1. **Note that when an exception starts, PC will jump to 0x4000000, which is the exception entrance. And software takes over the handling task from this point. First, the context of the CPU should be saved. Usually software should save all the registers and the stack.**
2. **After properly saving the context, software needs to know what the cause is of the occurring exception. The EXC (exception code) part of the Cause register offers the information.**
3. **If the cause of the occurring exception is an external interrupt, then check the interrupt pending bits of Cause register, and probably there will be more than one external interrupt pending. If that is true, software takes hold the priority management of the pending external interrupts. Attention should be paid that after servicing an external interrupt, the pending bit of Cause register must be cleared. If the occurring exception is not an external interrupt, then it’s internal. Like a clock tick timer interrupt, a system call, a range issue and so forth.**
4. **After finding out the cause of the occurring exception, the according ISR (Interrupt Service Routine) is called and the exception is handled.**
5. **Before leaving from an exception, software needs to know the original interrupted PC, which is offered by EPC register. And what is very important, the context needs to be restored properly before leaving.**

* The On-CPU Tick Timer

The scheme used in QS-I to implement an on-cpu tick timer is similar to the MIPS architecture, which implements two registers, Counter(#9) and Compare(#11). The Counter register is usually not modifiable by software, which is used to create a system clock. In other words, the Counter register increments itself by 1 every system clock cycle. The Compare register is R/W by software. Whenever the Counter is equal to the Compare, a tick timer exception occurs. Note that when a tick timer exception occurs, software needs to set a new value for the Compare register so as to get another tick timer exception. The code blow below is a way to set one tick timer trigger.

MFC0 $t0, Counter

ADDI $t0, $t0, 1000

MTC0 $t0, Compare

…