# Exceptions

Definitions:

In this document an *exception* means an event which will halt the processing of the current thread immediately and causes the core to switch to an exception handling routine. An exception is considered an error condition and has to be dealt with immediately. Note that very often in literature exception means interrupting the processor in general. See also interrupts.

*Table xx, Exception types and codes.*

| pri | code | name | description |
|---|---|---|---|
| 10 | 00000000 | instruction address violation [3] | While in user mode, instruction is fetched from memory address not allowed for user. |
| 6 | 00000001 | unknown opcode | Version 1.0 of COFFEE RISC does not have any unused opcodes which makes this obsolete. |
| 7 | 00000010 | Illegal instruction | While in 16 bit mode, trying to execute an instruction which is valid only in 32 bit mode or trying to execute a superuser only instruction in user mode. |
| 3 | 00000011 | miss aligned jump address [4] | Calculated jump target is not aligned to word(32 bit mode) or halfword(16 bit mode) boundary. |
| 2 | 00000100 | jump address overflow | A PC relative jump below the bottom of the memory or above the top of the memory. |
| 9 | 00000101 | miss aligned instruction address [1] | Instruction address is not aligned according to mode. This can be caused by:<br>- External boot address was not aligned to word boundary<br>- An interrupt vector is not properly aligned or interrupt mode is not correctly set<br>- Exception handler entry address is not aligned to word boundary (this will lock the core by causing an eternal loop!)<br>- System entry address is not aligned to word boundary |
| 8 | 111xxxxx | trap [2] | processor encountered a trap instruction |
| 5 | 00000110 | arithmetic overflow | The result of a signed arithmetic operation exceeds $2^{31}$-1 or falls below $-2^{31}$. |
| 0 | 00000111 | data address violation | While in user mode, a data address refers to memory address not allowed for user. |
| 1 | 00001000 | data address overflow | Trying to index data below of the bottom or above of the top of the memory |
| 4 | 00001001 | Illegal jump | Trying to jump to protected instruction memory area while in user -mode. |
| x | 00001010<br>...<br>00011111 | | Reserved for future extensions |

Notes:

[1] In this case, the address is saved, since it cannot be known which instruction(if any) caused the exception.
[2] For software exceptions (such as division by zero, or array bounds exceeded)
Exception address will point to trap –instruction. Note, that you cannot generate hardware exceptions using trap instruction because trap code will be padded with ones.
[3] If sequential execution traverses the boundary of the protected instruction memory area, the address of the instruction pointed to is saved.
[4] A jump between memory areas using different encoding will result in unpredictable behaviour.

**Handling an exception**

In case of an exception, core performs following tasks:

- Saves the address of the instruction causing the exception (or just an address, see table on previous page) to CCB register EXCEPTION_PC.
- Saves to CCB register EXCEPTION_PSR processor status flags which were used when the violating instruction was decoded .
- Saves the exception code (see table above) to CCB register EXCEPTION_CS.
- Disables interrupts.
- Switches to 32 bit decoding mode and superuser mode with register set 2 as default for reading and writing.
- Starts execution from a handler routine pointed by the CCB register EXCEP_ADDR.

Following things are guaranteed by hardware:

- The violating instruction is not able to modify the state of the processor (registers, status flags, data memory).
- All instructions before the violating one (in the order of execution) are executed.
- None of the instruction following the violating one are executed (pipeline is flushed up to the violating instruction).
- If multiple instructions on pipeline cause an exception simultaneously, the one which is first in the order of execution is taken into account.
- Interrupt requests cannot get through if an exception is signalled.
- An exception handler routine will allways see updated values of EXCEPTION_XX –registers immediately.

**Returning from the exception handler**

Depending on the handler, execution can be resumed from a different context or from the same context or it might not be resumed at all. In any case, appropriate flags should be written to SPSR (see registers) and the resume address should be written to PR31(the link register). Then, executing retu –instruction will update the PSR with flags written to SPSR and load the program counter with the value in PR31 causing the processor to start executing instructions from the desired memory location in the desired mode.

Notes:

- Remember to initialize EXCEP_ADDR –register appropriately in boot code. Incorrect address may cause eternal loop which will lock the processor until it is reset.
- Even though the violating instruction cannot change the contents of the memory, the address it refers to may appear on address bus.
- Interrupts are disabled when entering the handler routine but can be enabled by software (care must be taken).

- If the exception is caused by an interrupt service routine (see interrupts) and the routine is disabled permanently, you should pop the return address of that routine from the hardware stack to ensure correct operation of other interrupt routines. This is explained in the document interrupts.
- Exceptions are an inefficient way to interface superuser mode. Use scall – instruction instead of trap –instruction where appropriate.
-