Interrupts

8 different interrupt sources can be attached to the processor core. There is a fixed priority for all of the sources. There is a special register INT_PEND where interrupt requests are stored, so if one of the 8 interrupts occur, the request is stored in the INT_PEND register. The INT_PEND register is not visible to the programmer. Multiple interrupt requests may occur at the same time in the register, but only one interrupt at a time from a single source. The interrupts are served following the fixed priority. Priorities are listed in the table below. Interrupts can be disabled with the *di* instruction and enabled with the *ei* instruction.

An interrupt is acknowledged, when the corresponding interrupt line goes high and then goes low again. The interrupt signal must be high over a rising edge of the system clock signal in order to be recognised. Each pulse is interpreted as one interrupt, regardless of how long the signal is high. The interrupt service is started when the corresponding interrupt line goes low.

Interrupt priorities. Number 1 corresponds to the highest priority.

Priority(descending)	**Name**	**Notes**
1	coprocessor number 0 interrupt	not maskable
2	coprocessor number 1 interrupt	not maskable
3	coprocessor number 2 interrupt	not maskable
4	coprocessor number 3 interrupt	not maskable
5	Interrupt from the Int(0) input port	mask(0) input port
6	Interrupt from the Int(1) input port	mask(1) input port
7	Interrupt from the Int(2) input port	mask(2) input port
8	Interrupt from the Int(3) input port	mask(3) input port
9	Interrupt from the Int(4) input port	mask(4) input port
10	Interrupt from the Int(5) input port	mask(5) input port
11	Interrupt from the Int(6) input port	mask(6) input port
12	Interrupt from the Int(7) input port	mask(7) input port

If the same interrupt that is currently served, occurs during the interrupt service, new interrupt is not taken. After the current interrupt service has been finished, the new interrupt is served. On the contrary, interrupts with higher priority can interrupt the service of current interrupt, if interrupts are enabled. When entering the interrupt service routine, other interrupts are disabled by default. The user may enable interrupts during an interrupt service with the *ei* instruction.

Interrupts

There is a special stack for interrupts, where the return address is automatically stored. The program flow returns to the return address after the interrupt has been served. The processors state register PSR is also stored to the stack. The *reti* instruction loads the return address and the PSR from the stack. 8 addresses can be stored to the stack, so it can not overflow, since the interrupt service program can not be interrupted from the currently served source, and since there are 8 interrupt sources.

For example if the processor is currently serving an interrupt from the interrupt line number 4 and there occurs an interrupt at a higher priority, say at line 3, then the processor moves to serve the interrupt with the higher priority. (Of course, the higher priority interrupt, which occurred later, is served only if interrupts were enabled.)
If now during the service of interrupt number 3, the number 4 interrupt occurs again, it is not served, since it has lower priority. So there can not be a situation, where there would be multiple 'copies' of one interrupt waiting for service.

The interrupt logic has its own internal flags for the state of each interrupt. The *reti* instruction signals that the current interrupt has been served and if there is a new interrupt pending in the INT_PEND register, it is served immediately.

When starting to serve an interrupt, the processor automatically switches to 32 bit mode, regardless of the current mode. After serving an interrupt the processor returns to the mode it was before the interrupt occurred. (The PSR register is returned into its original state.)

Before an interrupt is actually served, the following happens:
1. The return address is stored into the hardware stack.
2. The state of the processor (the PSR register) is stored into the hardware stack.
3. A flag corresponding the interrupt is set to indicate that the interrupt is currently served and that later interrupts from the same source are not served.
4. The INT_MOD register is read to check whether the interrupt is to be served in the user mode or in the superuser mode.
5. The address in the register INTVi corresponding the interrupt is loaded and the interrupt service program is started in the 32 bit mode.

The *reti* instruction is used to return from the interrupt service program. The execution of the instruction does the following:
1. The interrupt is signaled as served to the interrupt service logic.
2. The logic checks if there are new interrupts waiting for service
3. If there are not any interrupt requests pending, the return address and the state of PSR register is loaded from the hardware stack.