Processor Operating Modes

16 bit mode and 32 bit decoding modes

16 bit mode refers to length of the instruction word. When in this mode, core expects to get instruction words encoded in 16 bits. Mode can be switched on the fly using *swm* –instruction. Of course when running actual code, the encoding really has to change after *swm* –instruction (See document instruction execution cycle times).

Limitations in 16 bit mode

- only 8 registers per set available: registers 24...31 mapped as registers 0...7
- Conditional execution is not available
- Only one condition register(CR0) in use
- Immediate constants are shorter, see instruction specifications.
- Instructions lui, lli, exbfi and cop not available (available as pseudo –operations if supported by assembler).
- 2nd source register and destination register shared.

Super user mode

The core can operate in *super user* –mode or *user* –mode. In super user –mode, core can access the whole memory space and both register banks. In user –mode, access to protected memory areas (software configurable) is denied and only 1st register bank is accessible. It's possible to switch from super user -mode to user –mode but not vice versa, except using scall –instruction which transfers execution to system code. System code entry address must be configured in startup code. Interrupt service routines can be run in both modes. This can also be configured by startup code. Core boots in super user –mode, which makes it possible to do the necessary configurations before starting application in user –mode.

Resetting the processor

After powering up the core, rst_x pin should be pulsed low (clock has to be stable) to set the core in correct state. If boot address selection is enabled (boot_sel –pin pulled high), boot address should be driven to data bus simultaneously with rst_x –signal. If boot address selection is disabled, core will boot at address 0x00000000h. Normal operation will start two clock cycles after the rising edge of the rst_x –signal. See document COFFEE_interface –about signal timing at reset.

Defaults after reset and boot procedure

Core will boot in super user and 32 bit –modes. Interrupts are disabled. A typical boot procedure would be to execute assembly written boot code which sets all CCB registers to suitable values and switches to user –mode by executing *retu* –instruction. See instruction specifications. See 'COFFEE_register' about reset values of configuration registers.

About configuring the core

Several features of the core can be configured via the core configuration block (CCB) which is a memory mapped register bank. When writing a new value to a configuration register, the new value will be valid when the instruction accessing CCB is in stage 5 of the pipeline. It follows that, if some configurations affect the execution of some instructions, or some configurations should be valid, when executing certain instructions, one has to make sure that there is enough instructions between the ones accessing CCB and dependent instructions. These can be nop –instructions or other instructions which do not depend on values of the configuration registers. Table below shows few examples of situations where it is essential to have few instructions between a CCB write and an instruction depending on the configuration made. If you're not sure about the number of 'guard' –instructions, use four.

instruction	purpose	notes	Dependency
st R1, R0, 0h	Remapping CCB to new address.	Assume R0 contains address of CCB_BASE – register and R1 contains a new address for CCB.	The 2nd st –instruction needs the value of CCB_BASE in stage 3 of the pipeline. CCB_BASE is valid when the 1st st –instruction is in stage 5 of the pipeline => There needs to be one instruction between the stores. In this case it is addi.
addi R0, R1, 1h	incrementing the new address of CCB. R0 should point now to CCB_END.	'guard' instruction	
st R2, R0, 0h	Configuring the size of configuration block itself (internal + external blocks)	Assuming R2 contains an address to be written to CCB_END.	
instruction	**purpose**	**notes**	**Dependency**
st R1, R0, 0h	Set an interrupt vector.	Assume R0 contains address of EXT_INT0_VEC and R1 points to interrupt service routine.	Interrupt vector will be valid when st –instruction has proceeded to stage 5 of the pipeline. Interrupts will be enabled when ei –instruction reaches stage 2 of the pipeline. Need to fill stages 3 and 4 to be safe.
nop	idle instructions ('guard' instructions)	Could use some other 'useful' instructions	
nop			
ei	Enable interrupts		
instruction	**purpose**	**notes**	**Dependency**
st R1, R0, 0h	Configure register translation for coprocessor access.	Assume R0 contains address of CREG_INDX_I and R1 valid configuration.	Configuration will be valid when st –instruction has proceeded to stage 5 of the pipeline. Configuration is needed when cop –instruction reaches stage 2 of the pipeline. Need to fill stages 3 and 4 to be safe.
nop	idle instructions ('guard' instructions)	Could use some other 'useful' instructions	
nop			
cop sqr(R2, R15)	Transfer an instruction word to coprocessor for execution		