Instruction specifications(draft)

General Information

This document describes the machine instructions implemented in COFFEE RISC 1. Descriptions are written in english. RTN (Register Transfer Notation) descriptions are available in appendix XX (will be available soon??) The followig set of instructions is the minimum set which every assembler should provide. With pseudo instructions the assembly language interface can be extended (A hint to implementer).

Abbreviations used

creg : condition register, number in the range 0...7
cond : condition (see table 'Condition codings')
dreg : destination register (32 bit mode) , number in the range 0...31
*sreg*i, *sreg* : source register (32 bit mode) , number in the range 0...31
dr : destination register (16 bit mode) , number in the range 0...7
*sr*i : source register (16 bit mode) , number in the range 0...7
imm, imm1, imm2 : immediate constant, see table 'Permitted values for immediate constants'
cp_sreg : coprocessor source register , number in the range 0...31
cp_dreg : coprocessor destination register , number in the range 0...31

Notes about instruction definitions:

16 bit mode refers to instruction word length. Data is manipulated in 32 bit words except with 16 bit multiplication instructions.
Syntax definition is an abstraction. The only purpose is to illustrate what an instruction expects as input and produces as output. The syntax of an assembly language program written for COFFEE RISC depends on the assembler and is documented in the respective assembler manual.
If the syntax of an instruction is different in 16 bit mode than in 32 bit mode then both syntaxes are presented: First the 32 bit version and then 16 bit version separated with a backslash. If both syntaxes are similar (or the particular instruction is not defined in 16 bit mode) then only one is presented.
Optional parameters for conditinal execution are enclosed in brackets.
Conditional execution is not allowed in 16 bit mode.
For timing of the instructions, see appendix XX(will be available soon?? You wish!).

Instruction definitions

add

syntax: **(cond, creg) add** dreg, sreg1, sreg2/ **add** dr, sr

description: The contents of the source registers *sregi* are summed together and the result is placed to the destination register *dreg*. Exception is generated if the result exceeds 2^{31}-1 or falls below -2^{31}. In 16 bit mode the register dr is the second source and the destination.

notes: Operation is carried out using twos complement arithmetics.

addi

syntax: **(cond, creg) addi** dreg, sreg1, imm/ **addi** dr, imm

description: The immediate constant is sign extended and summed with the contents of the source register *sreg1*. The result is placed to the destination register *dreg*. Exception is generated if the result exceeds 2^{31}-1 or falls below -2^{31}. In 16 bit mode the register dr is the first source register and the destination.

notes: Operation is carried out using twos complement arithmetics. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

addiu

syntax: **(cond, creg) addiu** dreg, sreg1, imm/ **addiu** dr, imm

description: The immediate constant is zero extended and summed with the contents of the source register *sreg1*. The result is placed to the destination register *dreg*. Overflow is ignored. In 16 bit mode the register dr is the first source register and the destination.

flags: Z, N, C (creg0)

notes: The register operand can also be negative even though the instruction is supposed to be '*add with immediate, unsigned operands*'. The only difference to addi is that possible overflow condition is ignored. In general addition procedure is exactly the same for both kinds of operands (2C or unsigned) only the result is interpreted differently (in this case by the programmer or compiler). Flags are set as expected when using 2C arithmetic. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

addu

syntax: **(cond, creg) addu** dreg, sreg1, sreg2/**addu** dr, sr

description: The contents of the source registers *sregi* are summed together and the result is placed to the destination register *dreg*.Overflow is ignored. In 16 bit mode the register dr is the second source and the destination.

flags: C, N, Z (CREG 0)

notes: Addition wider than 32 bits can be carried out as follows: Add the lower 32 bits with *addu* and add one to the upper 32 bits if carry was set in condition register creg0 as a result of the first addition. The register operands can also be negative even though the instruction is supposed to be '*add, unsigned operands*'. The only difference to add is that possible overflow condition is ignored. In general addition procedure is exactly the same for both kinds of operands (2C or unsigned) only the result is interpreted differently (in this case by the programmer or compiler). Flags are set as expected when using 2C arithmetic.

and

syntax: **(cond, creg) and** dreg, sreg1, sreg2/**and** dr, sr

description: Bitwise Boolean AND operation is performed to the contents of the source registers *sregi*. The result is placed to the destination register *dreg*.In 16 bit mode the register dr is the second source and the destination.

andi

syntax: **(cond, creg) andi** dreg, sreg1, imm/**andi** dr, imm

description: The immediate constant is zero extended. Bitwise Boolean AND operation is performed to the extended immediate and the contents of the source register *sreg1*. The result is placed to the destination register *dreg*. In 16 bit mode the register dr is the register source and the destination.

notes: See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

bc

syntax: **bc** creg, imm/**bc** imm

description: If the carry flag in the condition register *creg* is high, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

begt

syntax: **begt** creg, imm/**begt** imm

description: If the flags in the condition register *creg* indicate that the condition **eqt (**equal or greater than) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

belt

syntax: **belt** creg, imm/**belt** imm

description: If the flags in the condition register *creg* indicate that the condition **elt (**equal or less than) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

beq

syntax: **beq** creg, imm/**beq** imm

description: If the flags in the condition register *creg* indicate that the condition **eq (**equal) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

bgt

syntax: **bgt** creg, imm/**bgt** imm

description: If the flags in the condition register *creg* indicate that the condition **gt (**greater than) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

blt

syntax: **blt** creg, imm/**blt** imm

description: If the flags in the condition register *creg* indicate that the condition **lt (**less than) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

bne

syntax: **bne** creg, imm/**bne** imm

description: If the flags in the condition register *creg* indicate that the condition **ne (**not equal) is true, program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC. In 16 bit mode the condition register used is allways creg0

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The branch offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

chrs

syntax: **chrs** imm

description: Specifies which register set is used for reading or writing. The source register(s) and the destination register doesn't have to reside in the same set. The register sets to be used are specified by the immediate *imm* according to the following table:

imm	write	read
0 (00b)	set 1 (user set)	set 1 (user set)
1 (01b)	set 1 (user set)	set 2 (super user set)
2 (10b)	Set 2 (super user set)	set 1 (user set)
3 (11b)	Set 2 (super user set)	set 2 (super user set)

notes: When execution in the super user mode begins the default register set for reading and writing is the super user set (set 2). When returning back to the user mode the default register set is the user set (set 1). This command is allowed only in super user mode. An exception is generated on an attempt to use this command in user mode. As a result, the user cannot see the register set intended only for super user. Not allowed to be executed conditionally.

cmp

syntax: **cmp** creg, sreg1, sreg2/**cmp** sr1, sr2

description: The contents of the source registers *sregi/sri* are compared as if they were signed numbers. The operation is logically done by subtracting the contents of *sreg2/sr2* from the contents of *sreg1/sr1*. Flags N, Z and C are set or cleared accordingly and saved to the condition register *creg*. In 16 bit mode the condition register is allways creg0.

flags: N, Z, C

notes: The logical subtraction *sreg1- sreg2/sr1 - sr2* cannot over/underflow. This instruction cannot be executed contitionally.

cmpi

syntax: **cmpi** creg, sreg1, imm/**cmp**i sr, imm

description: The immediate constant *imm* is sign extended and compared to the contents of the source register *sreg1/sr1* as if they were signed numbers. The operation is logically done by subtracting the immediate *imm* from the contents of *sreg1/sr1*. Flags N, Z and C are set or cleared accordingly and saved to the condition register *creg*. In 16 bit mode the condition register is allways creg0.

flags: N, Z, C

notes: The logical subtraction *sreg1- imm/sr - imm* cannot over/underflow. This instruction cannot be executed contitionally. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

conb

syntax: (cond, creg) **conb** dreg, sreg1, sreg2/**conb** dr, sr

description: Concatenates the least signicant bytes from the source registers to form a halfword. The least significant byte from the register sreg1 becomes the most significant byte of the halfword and the least significant byte from the register sreg2 becomes the least significant byte of the halfword. The resulting halfword is saved to the destination register dreg. The upper halword of the result is filled with zeros. In 16 bit mode dr corresponds to the second source register sreg2 (and the destination) and sr corresponds to sreg1.

notes: Note that ordering of operands is different in 16 bit mode from that of 32 bit mode.

conh

syntax: (cond, creg) **conh** dreg, sreg1, sreg2/**conh** dr, sr

description: Concatenates the least signicant halfwords from the source registers to form a word. The least significant halfword from the register sreg1 becomes the most significant halfword of the word and the least significant halfword from the register sreg2 becomes the least significant halfword of the word. The resulting word is saved to the destination register dreg. In 16 bit mode dr corresponds to the second source register sreg2 (and the destination) and sr corresponds to sreg1.

notes: Note that ordering of operands is different in 16 bit mode from that of 32 bit mode.

cop

syntax: **cop** imm1, imm2 (Coprocessor Operation)

description: Moves the immediate *imm2*(instruction word of the coprocessor in question) to coprocessor number *imm1*. The immediate *imm1* specifies one of four possible coprocessors with values 0, 1, 2 or 3. The length of the *imm2* is 24 bits.

notes: Can be used only in 32 bit mode. This instruction cannot be executed contitionally. See coprocessor interface.

di

syntax: **di**

description: Disables maskable interrupts.

notes: Not permitted to be executed conditionally. See 'Interrupts and exceptions' for definitions and details

ei

syntax: **ei**

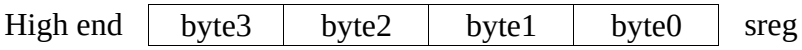
description: Enables maskable interrupts.

notes: Not permitted to be executed conditionally. See 'Interrupts and exceptions' for definitions and details

exb

syntax: (cond, creg) **exb** dreg, sreg, imm

description: Extracts the byte specified by the immediate *imm* from the source register *sreg/sr* and places it to the least significant end of the destination register *dreg/dr*. The upper three bytes in the destination register are cleared. The extracted byte is specified according to the following table.



imm	byte
0	byte0
1	byte1
2	byte2
3	byte3
other	undefined

exbf

syntax: (cond, creg) **exbf** dreg, sreg1, sreg2/**exbf** dr, sr

description: Operates like **exbfi**, but the eleven bits defining the extracted field are read from the least significant end of the source register *sreg2*. In the 16 bit mode *dr* is the second source and the destination.

notes: **Example.**
Suppose that a variable length bitfield should be extracted from register R0 (could be for example a sub address field in a message frame). Assume that the length of the bitfield of interest is contained in register R1 and the lsb position is in register R2. The following code could be used to extract the bitfield to R3:

```
slli  R2, R2, 6
or    R2, R2, R1
exbf  R3, R0, R2
```

See also **exbfi**.

exbfi

syntax: **exbfi** dreg, sreg1, imm

description: Extracts a bitfield of arbitrary length and position from the source register *sreg1* and places it to the low end of the destination register *dreg*. Bitfield length and position are defined by the immediate *imm* as follows: Six most significant bits of the immediate define the length of the bitfield as x, where x is an unsigned integer formed by those six bits. Five last significant bits of the immediate *imm* specify the LSB position of the extracted bitfield in the source register. The total length of the immediate is eleven bits. If the extracted bitfield is shorter than 32 bits, the extra bit positions in the destination register are filled with zeros.

notes: Can be used only in 32 bit mode. This instruction cannot be executed contitionally.

Example.

Extracting the field marked 'FIELD' from the source register:

	F	I	E	L	D										
...15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

x = 5 (000101) and LSB position = 10 (01010) => *imm* = 00010101010b = AAh

exh

syntax: (cond, creg) **exh** dreg, sreg1, imm

description: Extracts the halfword specified by the immediate *imm* from the source register *sreg1/sr* and places it to the least significant end of the destination register *dreg/dr*. The upper halfword in the destination register is cleared. If *imm* = 0, then the least significant halfword is extracted, otherwise the most significant halfword is extracted.

jal

syntax: **jal** imm

description: Program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter

PC. Link address is saved to register R31/SR31. The link address is the address of the next instrucion after branch slot.

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The jump offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

jalr

syntax: (cond, creg) **jalr** sreg1

description: Program execution branches to target address specified by the contents of the source register *sreg1/sr*. Link address is saved to register R31/SR31. The link address is the address of the next instrucion after branch slot.

notes: The instruction following this instrcution is always executed (branch slot). Conditional jumps (branches) which can reach the whole address space can be synthesized by executing this instruction conditinally.

jmp

syntax: **jmp** imm

description: Program execution branches to target address specified by the immediate *imm*. The target address is calculated as follows: The immediate offset *imm* is shifted left by one bit and sign extended. The sign extended offset is added to the contents of the program counter PC.

notes: This instruction cannot be executed contitionally. The instruction following this instrcution is always executed (branch slot). The jump offset is calculated relative to the instruction in the slot. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

jmpr

syntax: (cond, creg) **jmpr** sreg1

description: Program execution branches to target address specified by the contents of the source register *sreg1/sr*.

notes: The instruction following this instrcution is always executed (branch slot). Conditional jumps (branches) which can reach the whole address space can be synthesized by executing this instruction conditinally.

ld

syntax: (cond, creg) **ld** dreg, sreg1, imm

description: Loads a 32 bit data word from memory to the destination register *dreg/dr*. The address of the data is calculated as follows: The immediate offset *imm* is sign extended and added to the contents of the source register *sreg1/sr*. Two least significant bits of the resulting address are ignored and always driven low, so the data is expected to be aligned to word boundary.

notes: The result of the address calculation doesn't have to be aligned to word boundary. The two least significant bits can be used for example as byte index. See **exb** instruction. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

lli

syntax: **lli** dreg, imm

description: Loads the lower halfword of the destination register *dreg* with the immediate *imm*. The upper half of the destination register is cleared.

notes: Can be used only in 32 bit mode. This instruction cannot be executed contitionally. See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

lui

syntax: **lui** dreg, imm

description: Loads the upper halfword of the destination register *dreg* with the immediate *imm*. The lower half of the destination register is cleared.

notes: Can be used only in 32 bit mode. This instruction cannot be executed contitionally. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

mov

syntax: (cond, creg) **mov** dreg, sreg1

description: Copies the contents of the source register *sreg1/sr* to the destination register *dreg/dr*.

movcfc

syntax: (cond, creg) **movcfc** imm, dreg

description: Copies the contents of the status/control register of the coprocessor number *imm* to the destination register *dreg/dr*. The immediate *imm* is used to specify one of the four possible coprocessors: 0, 1, 2 or 3.

notes: This command is equivalent to **movdfc** imm, dreg, 0. See ‘Coprocessor Interface’.

movctc

syntax: (cond, creg) **movctc** imm, sreg1

description: Copies the contents of the source register *sreg1/sr* to the coprocessor control/status register. The immediate *imm* is used to specify one of the four possible coprocessors: 0, 1, 2 or 3.

notes: This command is equivalent to **movdtc** imm, 0, sreg1. See ‘Coprocessor Interface’.

movdfc

syntax: (cond, creg) **movdfc** imm, dreg, cp_sreg

description: Copies the contents of one of the registers in the coprocessor number *imm* to the destination register *dreg/dr*. The immediate *imm* is used to specify one of the four possible coprocessors: 0, 1, 2 or 3. *Cp_reg* is an index to the coprocessor register file.

notes: See 'Coprocessor Interface'.

movdtc

syntax: (cond, creg) **movdtc** imm, cp_dreg, sreg1

description: Copies the contents of the source register *sreg1/sr* to the coprocessor register *cp_dreg*. The immediate *imm* is used to specify one of the four possible coprocessors: 0, 1, 2 or 3.

notes: See 'Coprocessor Interface'.

mulhi

syntax: (cond, creg) **mulhi** dreg

description: Returns the upper 32 bits of a 64 bit product based on the previous instruction which has to be one of the instructions **mulu**, **muls**, **muli** or **mulus.**

notes: See also **mulu**, **muli, muls** ja **mulus**.

muli

syntax: (cond, creg) **muli** dreg, sreg1, imm/**muli** dr, imm

description: Multiplies the contents of the source register *sreg1* with the sign extended immediate *imm* and places the result to the destination register *dreg*. The operands are assumed to be signed integers (2C). In 16 bit mode *dr* is the source and the destination register.

notes: See **mulhi** for recovering the upper 32 bits of a product longer than 32 bits. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

muls

syntax: (cond, creg) **muls** dreg, sreg1, sreg2/**muls** dr, sr

description: Multiplies the contents of the source register *sreg1* with the source register *sreg2* and places the lower 32 bits of the result to the destination register *dreg*. The operands are assumed to be signed integers (2C). In 16 bit mode *dr* is the second source register and the destination.

notes: See **mulhi** for recovering the upper 32 bits of a product longer than 32 bits.

muls_16

syntax: (cond, creg) **muls_16** dreg, sreg1, sreg2/**muls_16** dr, sr

description: Multiplies the lower halfword of the source register *sreg1* with the lower halfword of the source register *sreg2* and places the result to the destination register *dreg*. The operands are assumed to be signed integers (2C). In 16 bit mode *dr* is the second source register and the destination.

mulu

syntax: (cond, creg) **mulu** dreg, sreg1, sreg2/**mulu** dr, sr

description: Multiplies the contents of the source register *sreg1* with the source register *sreg2* and places the lower 32 bits of the result to the destination register *dreg*. The operands are assumed to be unsigned integers). In 16 bit mode *dr* is the second source register and the destination.

notes: See **mulhi** for recovering the upper 32 bits of a product longer than 32 bits.

mulu_16

syntax: (cond, creg) **mulu_16** dreg, sreg1, sreg2/**mulu_16** dr, sr

description: Multiplies the lower halfword of the source register *sreg1* with the lower halfword of the source register *sreg2* and places the result to the destination register *dreg*. The operands are assumed to be unsigned integers. In 16 bit mode *dr* is the second source register and the destination.

mulus

syntax: (cond, creg) **mulus** dreg, sreg1, sreg2/**mulus** dr, sr

description: Multiplies the contents of the source register *sreg1* with the source register *sreg2* and places the lower 32 bits of the result to the destination register *dreg*. The operand in register *sreg1* is assumed to be an unsigned integer and the operand in register *sreg2* is assumed to be a signed integer. In 16 bit mode *dr* is the second source register and the destination.

notes: See **mulhi** for recovering the upper 32 bits of a product longer than 32 bits.

mulus_16

syntax: (cond, creg) **mulus_16** dreg, sreg1, sreg2/**mulus_16** dr, sr

description: Multiplies the lower halfword of the source register *sreg1* with the lower halfword of the source register *sreg2* and places the result to the destination register *dreg*. The operand in register *sreg1* is assumed to be an unsigned integer and the operand in register *sreg2* is assumed to be a signed integer. In 16 bit mode *dr* is the second source register and the destination.

nop

syntax: **nop**

description: Idle command that does not alter the state of the processor.

notes: See the list of instructions which require a succeeding nop. This instruction cannot be executed conditionally (even if it could it wouldn't have any effect anyway).

not

syntax: (cond, creg) **not** dreg, sreg1

description: Performs a bitwise Boolean NOT operation to the contents of the source register *sreg1/sr* and places the result to the destination register *dreg/dr*.

or

syntax: (cond, creg) **or** dreg, sreg1, sreg2/**or** dr, sr

description: Performs a bitwise Boolean OR operation to the contents of the source registers *sregi* and places the result to the destination register *dreg*. In 16 bit mode dr is the second source and the destination register.

ori

syntax: (cond, creg) **ori** dreg, sreg1, imm/ **ori** dr, imm

description: Performs a bitwise Boolean OR operation to the contents of the source register *sreg1* and zero extented immediate *imm*. The result is placed to the destination register *dreg*. In 16 bit mode dr is the source and the destination register.

notes: See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

rcon

syntax: **rcon** sreg1

description: Restores the contents of all the condition registers from the source register *sreg1*.

notes: This instruction is not allowed to be executed conditionally. See programming hints.

reti

syntax: **reti**

description: Used for returning from an interrupt service routine. Loads PC and PSR from the hardware stack.

notes: See programming hints. Not allowed to be executed conditionally. **The instrcution following reti allways has to be a nop!**

retu

syntax: **retu**

description: Used for returning or moving from system code/super user mode to user mode. Execution of user code starts from a address in register SR31. Status flags are copied from the register PSR_2. (They should be set appropriately before issuing retu).

notes: See scall. See programming hints. Not allowed to be executed conditionally. **The instrcution following retu allways has to be a nop!**

scall

syntax: (cond, creg) **scall**

description: System call transfers the processor to the super user mode and execution of instructions begins at address **sys_addr_c**. The link address is saved in to the register SR31. The link address is the address of the instruction following **nop** (see notes below). The state of the processor before scall is copied to the register PSR_2.

notes: When transferring the control to super user code the default settings are 32 bit mode, interrupts disabled and super user register set (both read and write). As with branches and jumps also this instruction has a branch slot which in this case has to be filled with a **nop** instruction. See **retu.** See also configuring the core before synthesis: configuration_pkg.vhd

scon

syntax: **scon** dreg

description: Saves the contents of all the condition registers to the (low end of) destination register *dreg.*

notes: This instruction is not allowed to be executed conditionally. See programming hints.

sext

syntax: (cond, creg) **sext** dreg, sreg1, sreg2/**sext** dr, sr

description: Works as **sexti**, but the position of the sign bit is calculated by using the five least significant bits from the source register *sreg2.* In 16 bit mode dr is the second source register and the destination.

notes: See also **sexti**.

sexti

syntax: (cond, creg) **sexti** dreg, sreg, imm/**sexti** dr, imm

description: Sign extends the operand in the source register *sreg* and places the result to the destination register *dreg*. The position of the sign bit is specified by the immediate *imm* (0 corresponds to LSB and 31 corresponds to MSB). In 16 bit mode *dr* is the source register and the destination.

notes: See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

sll

syntax: (cond, creg) **sll** dreg, sreg1, sreg2/**sll** dr sr

description: Performs the logical shift left to the contents of the source register *sreg1/sr* and places the result to the destination register *dreg/dr*. The six least significant bits in the source register *sreg2* specify the amount of shift. In 16 bit mode *dr* is the second source register and the destination.

notes: If the unsigned integer formed by the six least significant bits in the source register *sreg2* imply a shift of more than 32 positions then the result will be a shift of 32 positions (which is zero).

slli

syntax: (cond, creg) **slli** dreg, sreg1, imm/**slli** dr, imm

description: Performs the logical shift left to the contents of the source register *sreg1* and places the result to the destination register *dreg*. The immediate *imm* specifies the amount of shift. In 16 bit mode *dr* is the source register and the destination.

notes: See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

sra

syntax: (cond, creg) **sra** dreg, sreg1, sreg2/**sra** dr sr

description: Performs the arithmetic shift right to the contents of the source register *sreg1/sr* and places the result to the destination register *dreg/dr*. The six least significant bits in the source register *sreg2* specify the amount of shift. In 16 bit mode *dr* is the second source register and the destination.

notes: If the unsigned integer formed by the six least significant bits in the source register *sreg2* imply a shift of more than 32 positions then the result will be a shift of 32 positions.

srai

syntax: (cond, creg) **srai** dreg, sreg1, imm/**srai** dr, imm

description: Performs the arithmetic shift right to the contents of the source register *sreg1* and places the result to the destination register *dreg*. The immediate *imm* specifies the amount of shift. In 16 bit mode *dr* is the source register and the destination.

notes: See the permitted values for the immediate in the table 'Permitted values for immediate constants'.

srl

syntax: (cond, creg) **srl** dreg, sreg1, sreg2/**srl** dr sr

description: Performs the logical shift right to the contents of the source register *sreg1/sr* and places the result to the destination register *dreg/dr*. The six least significant bits in the source register *sreg2* specify the amount of shift. In 16 bit mode *dr* is the second source register and the destination.

notes: If the unsigned integer formed by the six least significant bits in the source register *sreg2* imply a shift of more than 32 positions then the result will be a shift of 32 positions.

srli

syntax: (cond, creg) **srli** dreg, sreg1, imm/**srli** dr, imm

description: Performs the logical shift right to the contents of the source register *sreg1* and places the result to the destination register *dreg*. The immediate *imm* specifies the amount of shift. In 16 bit mode *dr* is the source register and the destination.

notes: See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

st

syntax: (cond, creg) **st** sreg2, sreg1, imm

description: Stores the data in the source register *sreg2/sr2* to memory location whos address is calculated as follows: The immediate offset *imm* is sign extended and added to the contents of the source register *sreg1/sr1*.

notes: Two least significant bits of the data address are always driven low independent of the address calculation, so the data is aligned to word boundary. See **ld**. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’.

sub

syntax: (cond, creg) **sub** dreg, sreg1, sreg2/**sub** dr, sr

description: The contents of the source register *sreg2* is subtracted from the contents of the source register *sreg1* and the result is placed to the destination register *dreg*. Exception is generated if the result exceeds 2^{31}-1 or falls below -2^{31}. In 16 bit mode *dr* is the second source register and the destination.

notes: Operation is carried out using twos complement arithmetics

subu

syntax: (cond, creg) **subu** dreg, sreg1, sreg2/**subu** dr, sr

description: The contents of the source register *sreg2* is subtracted from the contents of the source register *sreg1* and the result is placed to the destination register *dreg*. In 16 bit mode *dr* is the second source register and the destination.

flags: Z, C, N

notes: Over/underflow is ignored. See programming hints

swm

syntax: **swm** imm

description: Changes the instruction length mode (16bit ? 32bit). The value of the immediate *imm* specifies the mode: *imm* = 16 => switch to 16bit mode, *imm* = 32 => switch to 32 bit mode.

flags: IL

notes: This instruction is not allowed to be executed conditionally. See the permitted values for the immediate in the table ‘Permitted values for immediate constants’. **The instrcution following swm allways has to be a nop!**

trap

syntax: **trap**

description: Generates a software trap. Execution is started at the address of exception handlind routine **excep_addr_c**. The address of the **trap** instruction is saved in the EPC register.

notes: This instruction is not allowed to be executed conditionally. See programming hints. This instruction could be used to fill unused memory to catch programs which ‘loose control’.

xor

syntax: (cond, creg) **xor** dreg, sreg1, sreg2/**xor** dr, sr

description: Performs a bitwise XOR operation to the contents of the source registers *sreg1* and *sreg2*. The result is placed to the destination register *dreg*. In 16 bit mode the bitwise xor is performed to the contents of *dr* and *sr* and the result is placed into *dr*.

instruction	**Permitted values for** *imm*			notes
	16 bit	32 bit		
		conditional	unconditional	
addi	$-2^6 \dots 2^6-1$	$-2^8 \dots 2^8-1$	$-2^{14} \dots 2^{14}-1$	
addiu	$0 \dots 2^7-1$	$0 \dots 2^9-1$	$0 \dots 2^{15}-1$	
andi	$0 \dots 2^7-1$	$0 \dots 2^9-1$	$0 \dots 2^{15}-1$	
bxx [1]	$-2^9 \dots 2^9-1$	-	$-2^{21} \dots 2^{21}-1$	
chrs	0...3	-	0..3	
cmpi	$-2^6 \dots 2^6-1$	-	$-2^{16} \dots 2^{16}-1$	
cop (imm1) [2]	-	-	0...3	Only 32 bit mode
exb	0...3	0..3	0...3	
exbfi	-	-	$0 \dots 2^{11}-1$	Only 32 bit mode
exh	0 or 1	0 or 1	0 or 1	
jal	$-2^9 \dots 2^9-1$	-	$-2^{24} \dots 2^{24}-1$	
jmp	$-2^9 \dots 2^9-1$	-	$-2^{24} \dots 2^{24}-1$	
ld	-8...7	$-2^8 \dots 2^8-1$	$-2^{14} \dots 2^{14}-1$	
lli	-	-	$0 \dots 2^{16}-1$ (or $-2^{15} \dots 2^{15}-1$)	Only 32 bit mode
lui	-	-	$0 \dots 2^{16}-1$ (or $-2^{15} \dots 2^{15}-1$)	Only 32 bit mode
movcfc/ movdfc [2]	0...3 (imm1)	0..3 (imm1)	0..3 (imm1)	
movctc/ movdtc [2]	0...3 (imm1)	0..3 (imm1)	0..3 (imm1)	
muli	$-2^6 \dots 2^6-1$	$-2^8 \dots 2^8-1$	$-2^{14} \dots 2^{14}-1$	
ori	$0 \dots 2^7-1$	$0 \dots 2^9-1$	$0 \dots 2^{15}-1$	
sexti	0...31	0...31	0...31	
slli	0...32	0...32	0...32	
srai	0...32	0...32	0...32	
srli	0...32	0...32	0...32	
st	-8...7	$-2^8 \dots 2^8-1$	$-2^{14} \dots 2^{14}-1$	
swm[3]	16 or 32	-	16 or 32	

Table 1, Permitted values for immediate constants.

[1] xx is one of the following: *c*, *blt*, *bne*, *bgt*, *beq*, *begt* or *belt*
[2] *imm1* is a value in the range 0...3. *Imm2* is an instruction word recognized by a coprocessor. See the definition of the instruction *cop*.
[3] Actually only one bit (bit with weight 32) is checked, so other values are also acceptable. It is recommended though that assembler only allows values in the table.