Alt_RISC VHDL op-code definitions for 24-bit instructions with three data sizes, 64 registers and 6-bit op-codes

```
-- For R = 0 or S = 0: the operand is zero rather than register 0 (via never writing to register 0 & all regs init'd to zero)
-- Prefix register loaded with 18-bit value by op_PFX, concatenated in front of N on next N/I or sNN instruction
-- Three register instructions use "1xxxxx" codes
-- 6-bit immediate instructions use "x1xxxx" codes
-- 12 & 18-bit immediate instructions use "00xxxx" codes
--              two operand Op-codes                    XXXXXX DDDDDD RRRRRR SSSSSS
constant op_ADD         : std_logic_vector(5 downto 0) := "100000";     -- R + S => D
constant op_SUB         : std_logic_vector(5 downto 0) := "100001";     -- R - S => D
constant op_ADC         : std_logic_vector(5 downto 0) := "100010";     -- R + S + carry => D
constant op_SBC         : std_logic_vector(5 downto 0) := "100011";     -- R - S + carry => D
constant op_AND         : std_logic_vector(5 downto 0) := "100100";     -- R and S => D
constant op_ANDC        : std_logic_vector(5 downto 0) := "100101";     -- R and not S => D
constant op_OR          : std_logic_vector(5 downto 0) := "100110";     -- R or S => D
constant op_XOR         : std_logic_vector(5 downto 0) := "100111";     -- R xor S => D
--constant op_MUL       : std_logic_vector(5 downto 0) := "101000";     -- R * S => D (not implemented)
--constant op_MULU  : std_logic_vector(5 downto 0) := "101001";         -- (R * S)>>24 => D (not implemented)
--constant op_MULUS   : std_logic_vector(5 downto 0) := "101010";       -- (R * S)>>24 => D (signed)(not implemented)
--constant op_DIV       : std_logic_vector(5 downto 0) := "101011";     -- start DR/S (not implemented)
--constant op_FADD      : std_logic_vector(5 downto 0) := "101100";     -- R + S => D (not implemented)
--constant op_FSUB      : std_logic_vector(5 downto 0) := "101101";     -- R - S => D (not implemented)
--constant op_FMUL      : std_logic_vector(5 downto 0) := "101110";     -- R * S => D (not implemented)
--constant op_FDIV      : std_logic_vector(5 downto 0) := "101111";     -- R / S => D (not implemented)
constant op_LD          : std_logic_vector(5 downto 0) := "000000";     -- mem(R + S) => D
constant op_ST          : std_logic_vector(5 downto 0) := "000001";     -- D => mem(R + S)
constant op_CALL        : std_logic_vector(5 downto 0) := "000010";     -- branch to R + S, PC+1 => D
constant op_JMPCC       : std_logic_vector(5 downto 0) := "000011";     -- if condition D true branch to R + S
constant op_LDB         : std_logic_vector(5 downto 0) := "001000";     -- mem(R + S) => D
constant op_STB         : std_logic_vector(5 downto 0) := "001001";     -- D => mem(R + S)
constant op_LDH         : std_logic_vector(5 downto 0) := "001010";     -- mem(R + S) => D
constant op_STH         : std_logic_vector(5 downto 0) := "001011";     -- D => mem(R + S)
constant op_LDSB        : std_logic_vector(5 downto 0) := "001100";     -- sign extended mem(R + S) => D
constant op_LDSH    : std_logic_vector(5 downto 0) := "001110";-- sign extended mem(R + S) => D
```

```
--          6-bit immediate Op-codes                        XXXXXX DDDDDD RRRRRR sNNNNN
constant op_ADDI        : std_logic_vector(5 downto 0) := "110000";       -- R + N => D
constant op_SUBI        : std_logic_vector(5 downto 0) := "110001";       -- R - N => D
constant op_ADCI        : std_logic_vector(5 downto 0) := "110010";       -- R + N + carry => D
constant op_SBCI        : std_logic_vector(5 downto 0) := "110011";       -- R - N + carry => D
constant op_ANDI        : std_logic_vector(5 downto 0) := "110100";       -- R and N => D
constant op_ANDCI       : std_logic_vector(5 downto 0) := "110101";       -- R and not N => D
constant op_ORI                 : std_logic_vector(5 downto 0) := "110110";       -- R or N => D
constant op_XORI        : std_logic_vector(5 downto 0) := "110111";       -- R xor N => D
--constant op_MULI      : std_logic_vector(5 downto 0) := "111000";       -- R * N => D (not implemented)
--constant op_MULUI     : std_logic_vector(5 downto 0) := "111001";       -- (R * N)>>24 => D (not implemented)
--constant op_MULUSI: std_logic_vector(5 downto 0) := "111010";         -- (R * N)>>24 => D (signed)(not implemented)
--constant op_DIVI      : std_logic_vector(5 downto 0) := "111011";       -- start DR/N (not implemented)
--constant op_FADDI     : std_logic_vector(5 downto 0) := "111100";       -- R + N => D (not implemented)
--constant op_FSUBI     : std_logic_vector(5 downto 0) := "111101";       -- R - N => D (not implemented)
--constant op_FMULI     : std_logic_vector(5 downto 0) := "111110";       -- R * N => D (not implemented)
--constant op_FDIVI     : std_logic_vector(5 downto 0) := "111111";       -- R / N => D (not implemented)
constant op_LDN             : std_logic_vector(5 downto 0) := "011000";   -- mem(R + N) => D
constant op_STN             : std_logic_vector(5 downto 0) := "011001";   -- D => mem(R + N)
constant op_LDBN        : std_logic_vector(5 downto 0) := "010000";       -- mem(R + N) => D
constant op_STBN        : std_logic_vector(5 downto 0) := "010001";       -- D => mem(R + N)
constant op_LDHN        : std_logic_vector(5 downto 0) := "010010";       -- mem(R + N) => D
constant op_STHN        : std_logic_vector(5 downto 0) := "010011";       -- D => mem(R + N)
constant op_LDSBN       : std_logic_vector(5 downto 0) := "010100";       --  sign extendedmem(R + N) => D
constant op_LDSHN   : std_logic_vector(5 downto 0) := "010110";           --  sign extendedmem(R + N) => D
constant op_CALLN       : std_logic_vector(5 downto 0) := "011010";       -- branch to R + N, PC+1 => D
constant op_JMPN        : std_logic_vector(5 downto 0) := "011010";       -- branch to R + N      (see CALLN)       convenience op-code, D=0
constant op_JMPCCN      : std_logic_vector(5 downto 0) := "011011";       -- if condition D true branch to R + N
constant op_INN             : std_logic_vector(5 downto 0) := "011100";   -- Input port R+N => D
constant op_OUTN        : std_logic_vector(5 downto 0) := "011101";       -- D => output port R+N
--          12-bit immediate Op-codes                       XXXXXX DDDDDD sNNNNN NNNNNN
constant op_CALLR       : std_logic_vector(5 downto 0) := "000100";       -- branch to PC+sNN, PC+1 => D
constant op_BR          : std_logic_vector(5 downto 0) := "000100";       -- call to PC+sNN (HLT if sNN=0, NOP if sNN=1) convenience op-code,
D=0
```

```
constant op_HLT          : std_logic_vector(23 downto 0) := "000100000000000000000000";       -- branch relative to self convenience op-code,
D=0
constant op_BRCC         : std_logic_vector(5 downto 0) := "000101";       -- if condition D true branch to PC+sNN
constant op_LDSI         : std_logic_vector(5 downto 0) := "000110";       -- sNN => D
--     18-bit immediate Op-code                          XXXXXX NNNNNN NNNNNN NNNNNN
constant op_PFX                    : std_logic_vector(5 downto 0) := "000111";       -- NNN => prefix reg (concatenates with next sN or sNN or
sNNN)

--         condition codes, always located in the D register field
--           uncompressed CCR is signs of the two adder inputs and sign of result, carry out & 24-bit ALU result
--           compressed CCR is overflow, carry, MSB, exp all 1s, exp all 0s, mant all 1s, mant all 0s, LSB
--  Boolean instructions do not change carry or overflow but do change MSB
--constant cc_A            : std_logic_vector(5 downto 0) := "000000";       -- always true
--constant cc_NOP                  : std_logic_vector(5 downto 0) := "000001";       -- always false
constant cc_Z    : std_logic_vector(5 downto 0) := "000010";       -- true if result zero
constant cc_NZ           : std_logic_vector(5 downto 0) := "000011";       -- true if result not zero
constant cc_CS           : std_logic_vector(5 downto 0) := "000100";       -- true if result carry set
constant cc_CC           : std_logic_vector(5 downto 0) := "000101";       -- true if result carry clear
constant cc_MI           : std_logic_vector(5 downto 0) := "000110";       -- true if result MSB set
constant cc_PL           : std_logic_vector(5 downto 0) := "000111";       -- true if result MSB clear
constant cc_VS           : std_logic_vector(5 downto 0) := "001000";       -- true if signed overflow
constant cc_VC           : std_logic_vector(5 downto 0) := "001001";       -- true if no signed overflow
constant cc_LE           : std_logic_vector(5 downto 0) := "001010";       -- true if result less than or equal
constant cc_GT           : std_logic_vector(5 downto 0) := "001011";       -- true if result greater than
constant cc_GE           : std_logic_vector(5 downto 0) := "001100";       -- true if result greater than or equal
constant cc_LT           : std_logic_vector(5 downto 0) := "001101";       -- true if result less than
constant cc_LS           : std_logic_vector(5 downto 0) := "001110";       -- true if unsigned result low or same
constant cc_HI   : std_logic_vector(5 downto 0) := "001111";       -- true if unsigned result high
--         non-traditional condition codes
constant cc_OD           : std_logic_vector(5 downto 0) := "010000";       -- true if result LSB set
constant cc_EV           : std_logic_vector(5 downto 0) := "010001";       -- true if result LSB clear
constant cc_1    : std_logic_vector(5 downto 0) := "010010";       -- true if result = 1
constant cc_N1           : std_logic_vector(5 downto 0) := "010011";       -- true if result not = 1
constant cc_M1           : std_logic_vector(5 downto 0) := "010100";       -- true if result = -1
```

```
constant cc_NM1         : std_logic_vector(5 downto 0) := "010101";     -- true if result not = -1
constant cc_M2          : std_logic_vector(5 downto 0) := "010110";     -- true if result = -2
constant cc_NM2         : std_logic_vector(5 downto 0) := "010111";     -- true if result not = -2
constant cc_01          : std_logic_vector(5 downto 0) := "011000";     -- true if result = zero or one
constant cc_N01         : std_logic_vector(5 downto 0) := "011001";     -- true if result not = zero or one
constant cc_0M1         : std_logic_vector(5 downto 0) := "011010";     -- true if result = zero or negative one
constant cc_N0M1        : std_logic_vector(5 downto 0) := "011011";     -- true if result not = zero or negative one
constant cc_01M1        : std_logic_vector(5 downto 0) := "011100";     -- true if result = zero, one or negative one
constant cc_N01M1       : std_logic_vector(5 downto 0) := "011101";     -- true if result not = zero, one or negative one
constant cc_01M12       : std_logic_vector(5 downto 0) := "011110";     -- true if result = zero, one, negative one or negative 2
constant cc_N01M12      : std_logic_vector(5 downto 0) := "011111";     -- true if result not = zero, one, negative one or negative 2
--          alternate condition code names
constant cc_EQ          : std_logic_vector(5 downto 0) := "000010";     -- true if result equal             (same encoding as cc_Z)
constant cc_NE          : std_logic_vector(5 downto 0) := "000011";     -- true if result not equal         (same encoding as cc_NZ)
constant cc_LO          : std_logic_vector(5 downto 0) := "000100";     -- true if unsigned result low      (same encoding as cc_CS)
constant cc_HS          : std_logic_vector(5 downto 0) := "000101";     -- true if unsigned result high or same (same encoding as cc_CC)
```