```
////////////////////////////////////////////////////////////////////////
/////
// Opus16 CPU
// ISA : Instruction Set Architecture
//----------------------------------
// Instructions are all lower case, check syntax
// example:
// addrp r5,pD; // add register r5 the value of pointer D
// a..f/A..F same for regs and pointers
// semicolon is an instruction terminator
// comments start after //
//----------------------------------
////////////////////////////////////////////////////////////////////////
/////
// Notes:
// mark with * when register 0 is modified by any other instruction
// r0 (register 0) is not used after initialization
// r0 is a temporary register in operations involved
//    inmediate values in arithmetic and logic
//    register - pointer
//    bit test or swap
//
// Counters:
// dcjz and dcjnz are always executed once
// the counter c0 to c3 should -1 of the expected loop count
// example, for a 8 loop the c# value is 7
//
////////////////////////////////////////////////////////////////////////
/////
// Main defines
define  RESET_ADDR      'h0000 // Reset vector address, always address 0
define  MAIN_ADDR       'h0010 // Main entry address
define  BOOT_ADDR       'h3F00 // Boot loader address
define  STACK_LENGTH        64 //


   ////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////
   // Program Section, Protected area
   ////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////
@RESET_ADDR // Reset vector, fixed from 0 to 8
reset_v:    jmp   main          ; // @'h000 reset vector
irq1_v:     jmp   hw_irq1       ; // @'h002 hardware irq vector
irq2_v:     jmp   hw_irq2       ; // @'h004 hardware irq vector
irq3_v:     jmp   hw_irq3       ; // @'h006 hardware irq vector
irq4_v:     jmp   hw_irq4       ; // @'h008 hardware irq vector

// Free from h000A to MAIN_ADDR

@MAIN_ADDR
main:       nop                 ; // no operation
            ldpv    pF,stack    ; // initialize stack pointer
```

```
            // ARITHMETIC
lb_arith:   add     r1,r2       ; // r1 = r1 + r2
            adc     r3,r4       ; // r3 = r3 + r4 + CY
            addv    r5,'h1234   ; // r5 = 1234h                  *r0
            addrp   r6,p0       ; // r6 = (p0)                   *r0
            sub     r7,r8       ; // r7 = r7 - r8
            sbb     r9,ra       ; // r9 = r9 - ra - ~CY
            subv    rB,1        ; // rB = rB - 1                 *r0
            subrp   rC,p1       ; // rC = rC - p1                *r0

            // Logic
lb_logic:   not     rD          ; // rD = ~rD
            not     rD,r1       ; // r1 = ~rD
            and     rE,rF       ; // rE = rE and rF
            andv    r1,'hffff   ; // r1 = r1 and 'hffff          *r0
            or      r2,r1       ; // r2 = r2 or r1
            orv     r3,'h1010   ; // r3 = r3 or 10 hex           *r0
            xor     r4,r5       ; // r4 = r4 xor r5
            xorv    r6,'h1010   ; // r6 = r6 xor 10 hex          *r0
            inc     r7          ; // r7++
            dec     r8          ; // r8--
            cmpr    r9,rA       ; // r9-rA, set status flags
            cmprv   rb,'h0000   ; // rB-0, set status flags      *r0
            cmprp   rc,p2       ; // rC-p2,set status flags      *r0
            shl     r1          ; // r1<<1
            shr     r2          ; // r2>>1
            shl4    r3          ; // r3<<4
            shr4    r3          ; // r3>>4

            // MOVE
lb_move:    mvrr    r3,r4       ; // r4 - r3
            mvrp    r4,p4       ; // p4 = r4
            mvpr    p4,r7       ; // r7 = p4
            mvpp    p4,p5       ; // p5 = p4
            swap    r3,r4       ; // r4=r3, r3=r4
            swapp   r4,p3       ; // p3=r4, r4=p3                *r0

            // LOAD
lb_load:    ldcv    c0,3        ; // counter #0 = 3
            ldc     c1,r9       ; // counter #1 = 9
            ldr     r1,temp0    ; // r1 = (temp0)
            ldrv    r2,3        ; // r2 = 3
            ldrp    r3,p1       ; // r3 = (p1)
            ldrx    r4,p1,2     ; // r4 = (p1+2)
            ldrpi   r5,p2       ; // r5 = (p2), p2++
            ldrpd   r6,p3       ; // --p3, r6 = (p3)
            pull    r7,p4       ; // --p4, r7 = (p4)
            pop     r8,p5       ; // --p5, r8 = (p5)
            ldmam   m2          ; // internal
            ldpagv  'haba       ; // page = 'h0aba
            ldpag   r9          ; // page = r9

            // store group
lb_store:   str     r6,temp0    ; // (temp0) = r6
```

```
            strp    r1,p4       ; // (p4) = r1
            strx    r2,p4,2     ; // (p4+2) = r2
            strpi   r3,p5       ; // (p5) = r3, p5++
            push    ra,p8       ; // (p8) = ra, p8++
            strpd   rB,p9       ; // --p9, (p9) = rb

            // pointer group
lb_pointer: ldp     pa,tempa    ; // pa = (tempa)
            ldpv    pb,temp0    ; // pB = &temp0
            ldpy    pc,rb       ; // pc = (pc+rb)
            stp     pd,tempa    ; // (tempa) = pd
            incp    pE          ; // pE++
            decp    pe          ; // pE--

            // I/O group
lb_io:      inp     r1,port1    ; // r1 = (port1)
            inpp    r2,pc       ; // r2 = (pC)
            outp    r3,port2    ; // (port2) = r3
            outpp   r4,pd       ; // (pD) = r4

            // control transfer
main1:      bra     main1       ; // branch to main

            // Jump Group, full address range
main2:      dcjz    c0,main2    ; // if c0=0 jump main2
main3:      dcjnz   c1,main3    ; // if c1<>0 jump main3
            jmp     main4       ; // always jump main4
main4:      jmp     f1,main5    ; // if uflag 1 = 0 jump main5
main5:      jmp     tA,main6    ; // if uflag a = 1 jump main6
main6:      jmpp    pe          ; // jump to (pe), address is the
contents of pE
main7:      jsr     subr1       ; // jump subr1, direct address
subr1:      rts                 ; // return from subr1
            rti                 ; // return from interrupt

            // User flags and single bit output port.
lb_bits:    uflag   tE          ; // user flag E = 1
            uflag   ff          ; // user flag F = 0
            uport   ta          ; // oport a (10dec) = 1
            uport   fB          ; // oport b (11dec) = 0

            // Bit test
            bit     r1,r2       ; // r1 and r2, cc affected
            bitv    r3,'h0011   ; // r3 and  3, cc affected        *r0

            // Interrupts
lb_irq:     cli                 ; // disable interrupts
            sei                 ; // enable interruts
swia:       swi     swia        ; // software interrupt / trap

            // Special intructions
//          hwi                 ; // do not use it
            stop                ; // output stop signal
```

```
    ////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
    // Interrupt vectors
    ////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
            ////////////////////////
            // Interrupt #1
hw_irq1:    uport   t8              ; // use with scope
            uport   f8              ;
            rti                     ;

            ////////////////////////
            // Interrupt #2
hw_irq2:    uport   t9              ; // for time
            uport   f9              ;
            rti                     ;

            ////////////////////////
            // Interrupt #3
hw_irq3:    uport   tA              ; // measurement
            uport   fA              ;
            rti                     ;

            ////////////////////////
            // Interrupt #4
hw_irq4:    uport   te              ; // or logic analyzer
            uport   fE              ;
            rti                     ;

   ////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////
   // Data Section, Unprotected area
   ////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////
            // temporary data storage
temp0:      dw  33      ; // one word and initialize with 33 decimal
tempa:      ds  5       ; // reserve 5 words, uninitialized

   ////////////////////////////////////////////////////////////////////
   // port are real I/O address in HW
   // here is for simulation only
port1:      dw  'h0100  ; // address port1
port2:      dw  'h0108  ; // address port2
```