```
////////////////////////////////////////////////////////////////////////////
// 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, ra is same as rA
// semicolon is an instruction terminator
// comments start after // (forward slash)
//---------------------------------
////////////////////////////////////////////////////////////////////////////
// rn   register number (rd or rs)
// rd   register destination
// rs   register source
// pn   pointer  number (pd or ps)
// pd   pointer  destination
// ps   pointer  source
// pt   pointer
// val  inmediate value, always 16 bits
// type shift mode, left/right/rotate
// cn   counter number, c0 to c3 used by dcjnz/dcjz
// addr content of the address
// mn   memory access mode, n=0 to 3, look end of the page
// port I/O port number
// []   optional argument
// t/f  true/false for user flags and output flags
// #    flag number from 0 to f
////////////////////////////////////////////////////////////////////////////
                      Clocks  Machine
Mnemo  OpCode  Words  Cycles  Cycles    Syntax              // Description
                                        mnemo   arg1,arg2;  // Comments
////////////////////////////////////////////////////////////////////////////
// General Group
NOP     00      1       4       1       nop;                // no operation

////////////////////////////////////////////////////////////////////////////
// Arithmetic group
ADD     01      1       4       1       add     rd,rs;      // rd = rd + rs
ADC     02      1       4       1       adc     rd,rs;      // rd = rd + rs + cy
ADDV    03      2       8       2       addv    rd,val;     // rd = rd + val
ADDRP   04      1       4       1       addrp   rd,ps;      // rd = rd + ps
SUB     05      1       4       1       sub     rd,rs;      // rd = rd - rs
SBB     06      1       4       1       sbb     rd,rs;      // rd = rd - rs - ~cy
SUBV    07      2       8       2       subv    rd,val;     // rd = rd - val
SUBRP   08      1       4       1       subrp   rd,ps;      // rd = rd - ps

////////////////////////////////////////////////////////////////////////////
// Logic group
NOT     09      1       4       1       not     [rs,]rd;    // rd = ~rs
AND     0a      1       4       1       and     rd,rs;      // rd = rd & rs
ANDV    0b      2       8       2       andv    rd,val;     // rd = rd & val
OR      0c      1       4       1       or      rd,rs;      // rd = rd | rs
ORV     0d      2       8       2       orv     rd,val;     // rd = rd | val
XOR     0e      1       4       1       xor     rd,rs;      // rd = rd ^ rs
XORV    0f      2       8       2       xorv    rd,val;     // rd = rd ^ val
INC     10      1       4       1       inc     rd;         // rd = rd + 1
DEC     11      1       4       1       dec     rd;         // rd = rd - 1
CMPR    12      1       4       1       cmpr    rd,rs;      // rd - rs
CMPRV   13      2       8       2       cmprv   rn,val;     // rn - val
```

```
CMPRP   14      1       4       1       cmprp   rn,pt;      // rn - pt
SHL     15      1       4       1       shl     type,rn;    // rd = rd << 1
SHR     16      1       4       1       shr     type,rn;    // rd = rd >> 1
SHL4    17      1       4       1       shl4    rn;         // rd = rd << 4
SHR4    18      1       4       1       shr4    rn;         // rd = rd >> 4

////////////////////////////////////////////////////////////////////////
// Move Group
MVRR    19      1       4       1       mvrr    rs,rd;      // rd = rs
MVRP    1a      1       8       2       mvrp    rs,pd;      // pd = rs
MVPR    1b      1       4       1       mvpr    ps,rd;      // rd = ps
MVPP    1c      1       8       2       mvpp    ps,pd;      // pd = ps
SWAP    1d      1       4       1       swap    rn,rn;      // rn <--> rn
SWAPP   1e      1       8       2       swap    rn,pt;      // rn <--> pt

////////////////////////////////////////////////////////////////////////
// Load Group
LDCV    1f      2       8       2       ldcv    cn,val;     // c[0/1/2/3] = val
LDC     43      2       4       1       ldc     cn,rs;      // c[0/1/2/3] = reg
LDR     20      2       12      3       ldr     rd,addr;    // rd = (addr)
LDRV    21      2       8       2       ldrv    rd,val;     // rd = val
LDRP    22      1       8       2       ldrp    rd,ps;      // rd = (ps)
LDRX    23      2       12      3       ldrx    rd,ps,val;  // rd = (ps + val)
LDRPI   24      1       12      3       drpi    rd,ps;      // rd = (ps) => ps++
LDRPD   25      1       12      3       drpd    rd,ps;      // --ps, rd = (ps)
PULL    25      1       12      3       ldrpd   rd,ps;      // --ps, rd = (ps)
POP     25      1       12      3       ldrpd   rd,ps;      // --ps, rd = (ps)
LDMAM   26      1       8       2       ldmam   mn;         // m[0/1/2/3] access mode
LDPAGV  27      2       8       2       ldpagv  val;        // page address = value
LDPAG   42      1       4       1       ldpag   rs;         // page address = reg

////////////////////////////////////////////////////////////////////////
// Store Group
STR     28      2       12      3       str     rs,addr;    // (addr) = rs
STRP    29      1       8       2       strp    rs,pd;      // (pd) = rs
STRX    2a      2       12      3       strx    rs,pd,val;  // (pd + val) = rs
STRPI   2b      1       12      3       strpi   rs,pd;      // (pd) = rs => pd++
PUSH    2b      1       12      3       strpi   rs,pd;      // (pd) = rs => pd++
STRPD   2c      1       12      3       strpd   rs,pd;      // --pd, (pd) = rs

////////////////////////////////////////////////////////////////////////
// Pointer Group
LDP     2d      2       12      3       ldp     pd,addr;    // pd = (addr)
LDPV    2e      2       8       2       ldpv    pd,val;     // pd = val
LDPY    44      1       16      4       ldpy    pd,rs;      // pd = (pd+rs)
STP     2f      2       12      3       stp     ps,addr;    // (addr) = ps
INCP    30      1       4       1       incp    pt;         // pd = pd + 1
DECP    31      1       4       1       decp    pt;         // pd = pd - 1

////////////////////////////////////////////////////////////////////////
// I/O Group
INP     32      2       12      3       inp     rd,port;    //     rd = (port) => i/o
INPP    33      1       12      3       inpp    rd,ps;      //     rd = (ps)   => i/o
OUTP    34      2       12      3       outp    rs,port;    // (port) =  rs    => i/o
OUTPP   35      1       12      3       outpp   rs,pd;      //   (pd) =  rs    => i/o
//      36  // see DCJZ

////////////////////////////////////////////////////////////////////////
// Control Transfer : Branch Group, two's complements -2^15-1 to +2^15
BRA     37      2       8       2       bra     cond,addr;  // pc = pc + offset
```

```
////////////////////////////////////////////////////////////////////////
// Jump Group:  full address range
DCJZ    36      2       8       2       dcjz    cn,addr;    // pc = pc + offset,cn=0
DCJNZ   38      2       8       2       dcjnz   cn,addr;    // pc = pc + offset,cn~=0
JMP     39      2       8       2       jmp     [t/f]#,addr;// pc = addr
JMPP    3a      1       8       2       jmpp    pt;         // pc = pt
JSR     3b      2       12      3       jsr     addr;       // pc = addr    => --sp
RTS     3c      1       16      4       rts;                // pc = (sp)    => sp++
RTI     3d      1       20      5       rti;                // pc = (sp)    => sp++

////////////////////////////////////////////////////////////////////////
// Bit control
UFLAG   3e      1       4       1       uflag   [t/f]#;     // uflag bit n clr/set
UPORT   3f      1       4       1       uport   [t/f]#;     // oport bit n clr/set
// Test bits
BIT     40      1       4       1       bit     rd,rs;      // rd = rd & rs
BITV    41      2       8       2       bitv    rd,val;     // rd = rd & val
//      42      // see LDPAG
//      43      // see LDC
//      44      // see LDPY
SEI     45      1       4       1       sei;                // set interrupt,
enable
CLI     46      1       4       1       cli;                // clear interrupt,
disable
SWI     47      2       12      1       swi     addr;       // pc = (sp)    => sp++ 1

////////////////////////////////////////////////////////////////////////
// Special Instructions
HWI     5a      1       20      1       hwi ;               // hw interrupt, do not
use
STOP    ff      1       4       1       stop;               // hw signal, single bit

////////////////////////////////////////////////////////////////////////
// Conditional and mofifier fields.
////////////////////////////////////////////////////////////////////////
// conditional for branchs.
  z   1  // bra  z,label
  nz  9  // bra nz,label
  e   1  // bra  e,label
  ne  9  // bra ne,label
  c   2  // bra  c,label
  nc  a  // bra nc,label
  lo  2  // bra lo,label (unsigned)
  hs  a  // bra hs,label (unsigned)
  s   3  // bra  s,label
  ns  b  // bra ns,label
  o   4  // bra  o,label
  no  c  // bra no,label
  ge  5  // bra ge,label
  lt  d  // bra lt,label
  gt  6  // bra gt,label
  le  e  // bra le,label
  hi  7  // bra hi,label (unsigned)
  ls  f  // bra ls,label (unsigned)

// modifier for shifts
   l  0  // shr l,rd;   // shift left or right logical
   k  1  // shl k,rd;   // shift left or right link bit
   a  2  // shr a,rd;   // shift right arithmetic
```

```
   r  3  // shl r,rd;   // rotate left or right, link bit = msb or lsb.


// Shift modes:
    normal shifts:
    shl l,r#; (shift left  logical,   bit  0 = 0  , link_bit = msb)
    shr l,r#; (shift right logical,   bit 11 = 0  , link_bit = lsb)
    shr a,r#; (shift right arithmetic, bit 11 = msb, link_bit = lsb)

    link shifts:
    shl k,r#; (shift left  logical,   bit  0 = link_bit, link_bit = msb)
    shr k,r#; (shift right logical,   bit 11 = link_bit, link_bit = lsb)

    rotates:
    shl r#,r#;(shift left  logical,   bit  0 = msb, link_bit = msb)
    shr r#,r#;(shift right logical,   bit 11 = lsb, link_bit = lsb)

    undefined combinations:
    shl a,r#;

////////////////////////////////////////////////////////////////////////////
// Load Memory Access Mode : LDMAM
// Load and Store instructions from/to internal/external memory
        LDMAM m0; // store internal, load internal   0  0 ii
        LDMAM m1; // store internal, load external   0  1 ie
        LDMAM m2; // store external, load internal   1  0 ei
        LDMAM m3; // store external, load external   1  1 ee
////////////////////////////////////////////////////////////////////////////
```