Description of the nCore processor core
Ver. 0.2

There are 16 instructions. Every instruction takes 1 argument. As all the instruction is coded on 8 bits, for the arguments there are only 4 bits, and they could code a register or a 4-bits constant. There are many possibility to coding on more bits, but with this solution the decoding and execution is more simple and more fast.

There are also 16 registers. All of that is free for use.

There are no pipelines, because of a test. I've wrote it, but as an other core ran with ~40 MHz w/o, and ~60 MHz with it, it's better chose to build in two core rather than implementing one with two pipelines.

The execution time is always 1 clock.

The execution scheme is the following:

```
      A           B
      |           |
      L--------------
      |           L-- calculating the result of the addition
      |           |
      L--------------
      |           L-- calculating the result of the subtraction
      |           |
      L--------------
      |           L-- calculating the result of the bitwise and
      |           |
      .           .
      .           .
      .           .
```

So all the ALU instruction are calculated parallel. There are two register determining the results:
'A' and 'B'. We can modify the registries with the following instructions:

inst_mvA	: load the given register in the register 'A'
inst_mvB	: load the given register in the register 'B'
inst_coA	: load the given constant in the register 'A'
inst_coB	: load the given constant in the register 'B'

We can choose the destination, and the result to be stored, the target is always a register, coded the same way: 4 bits on the last significant side.

inst_and	: bitwise and:	a&&b
inst_orr	: bitwise or:	a||b
inst_xor	: bitwise xor:	a^^b
inst_shl	: shift left:	a<<b[3:0]
inst_shr	: shift right:	a>>b[3:0]
inst_add	: addition:	a+b
inst_sub	: subtraction:	a-b

The 4 last instruction may produced bits, that could not be coded on the target registers. These are implemented in the FLAG register.

FLAG[0]: zero, the result of the executed ALU instruction is 0.

FLAG[1]=the carry bit of the addition
FLAG[2]=the carry bit of the subtraction
FLAG[3]=the most significant bit of the shift left
FLAG[4]=the least significant bit of the shift left

After that, we have all the sources to calculating for ex. an addition.

If we would calculating the 11+3, and place the result in the register 8:

```
inst_coA 11       ;we load 11 in 'A'
inst_coB 3        ;we load 3 in 'B'
inst_add 8        ;we store the result in the register 8
```

After these instruction we need only one instruction to store 11-3 in the register 10:

```
inst_sub 8        ;we store the result in the register 10
```

because all the source registers are set with the good values.

There is 2 special registers. These are the FLAG, witch stores results
of the ALU instructions, DP, witch determines the address on the data memory
the 'data', what can use to overwrite or read the content of the data memory,
and the IP, the instruction pointer on the instruction memory.

```
      inst_Fmv : write the contents of the FLAG to the specified register
      (FLAG)
      inst_mvD : write the contents of the specified register in the data memory
      ('data')
      inst_Dmv : read  the contents of the specified register from the data memory
      ('data')
      inst_mvP : write the contents of the specified register in the data memory pointer
      (DP)

      inst_jmp : if the least significant bit of the register 'A' is set, jump to the
                  location determined in the specified register
      (IP)
```

As the registers are 16 bits wide, max the address space for the instructions is 64k*
8bits, for the data's 64k*16bits.