Università della Svizzera italiana

Faculty of Informatics

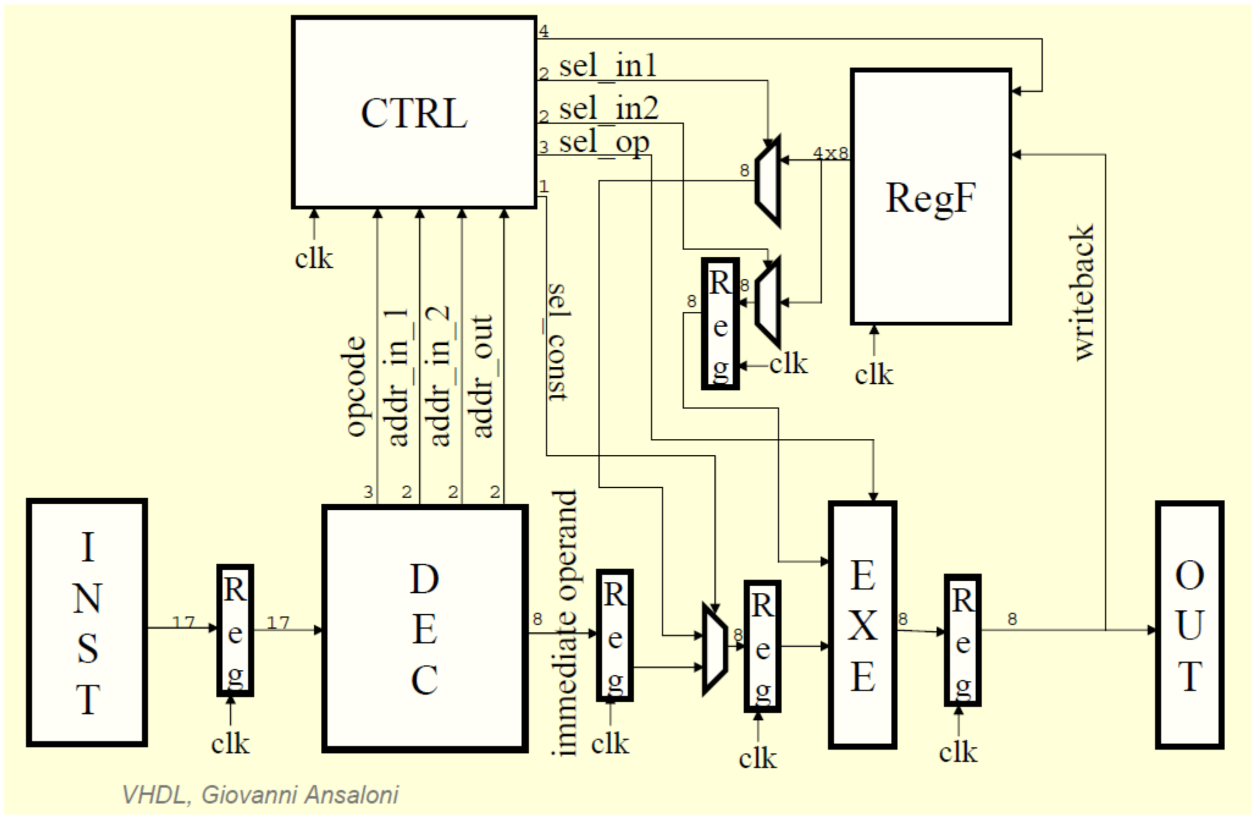
Advanced Learning and Research Institute

Umberto Bondi

Pablo Di Giulio

FINAL ASSIGNMENT
8-bit Processor

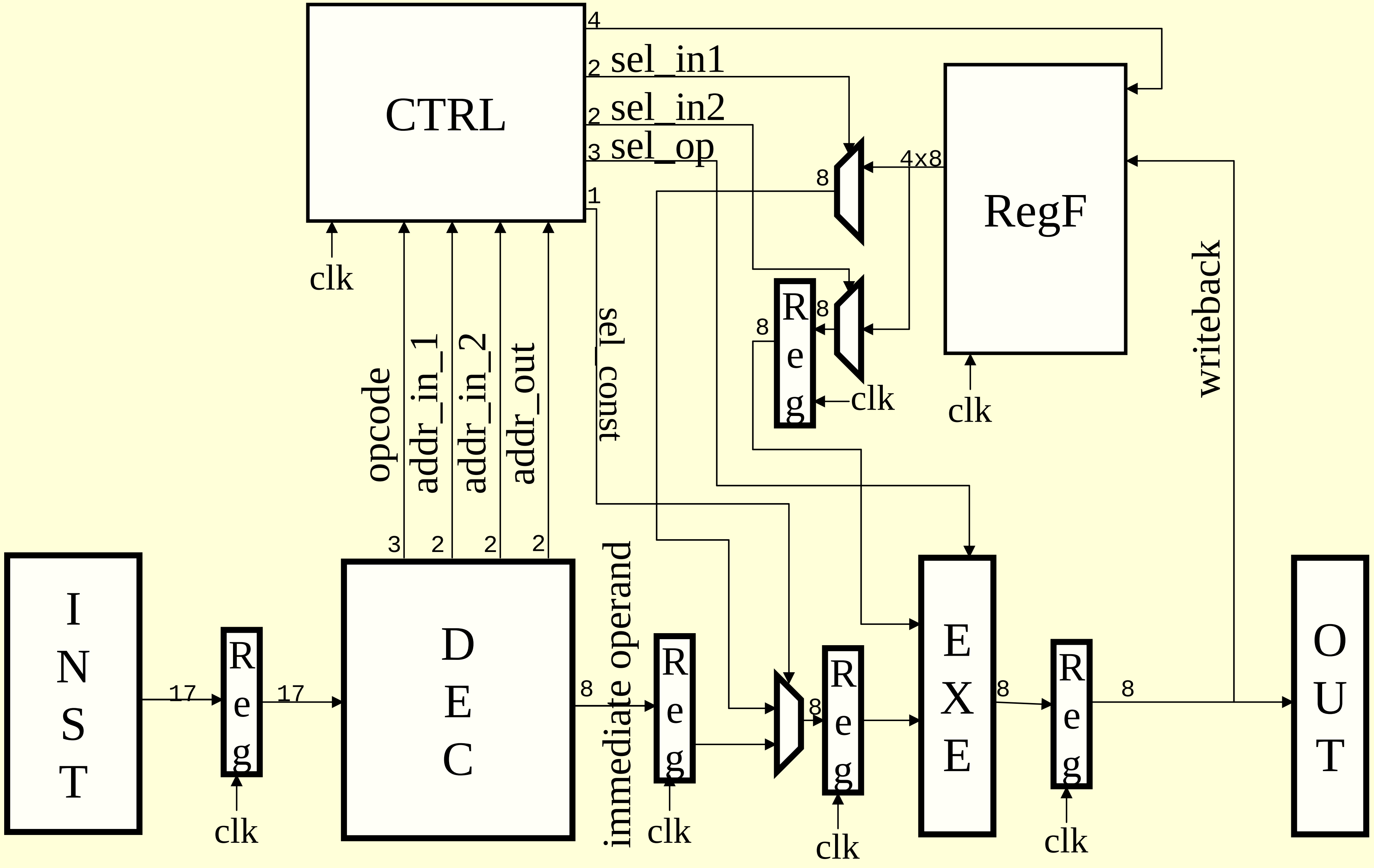
- Goal of the projects is to test and improve your VHDL knowledge through a real application.
- Design and test a 8-bit microprocessor as the following figure shows



- To test the correct functionality of the processor, a program that solves the following operation should be performed:
  - ✓ (0x5 * 0x3) + (0x8 * 0x2) = 0x1F
- This expression must be written using shifts and adds to adapt it to our instruction set:
  - ✓ (0x5 * 0x3) + (0x8 * 0x2) = (0x5 + (0x5 SL 1)) + (0x8 SL 1) = 0x1F

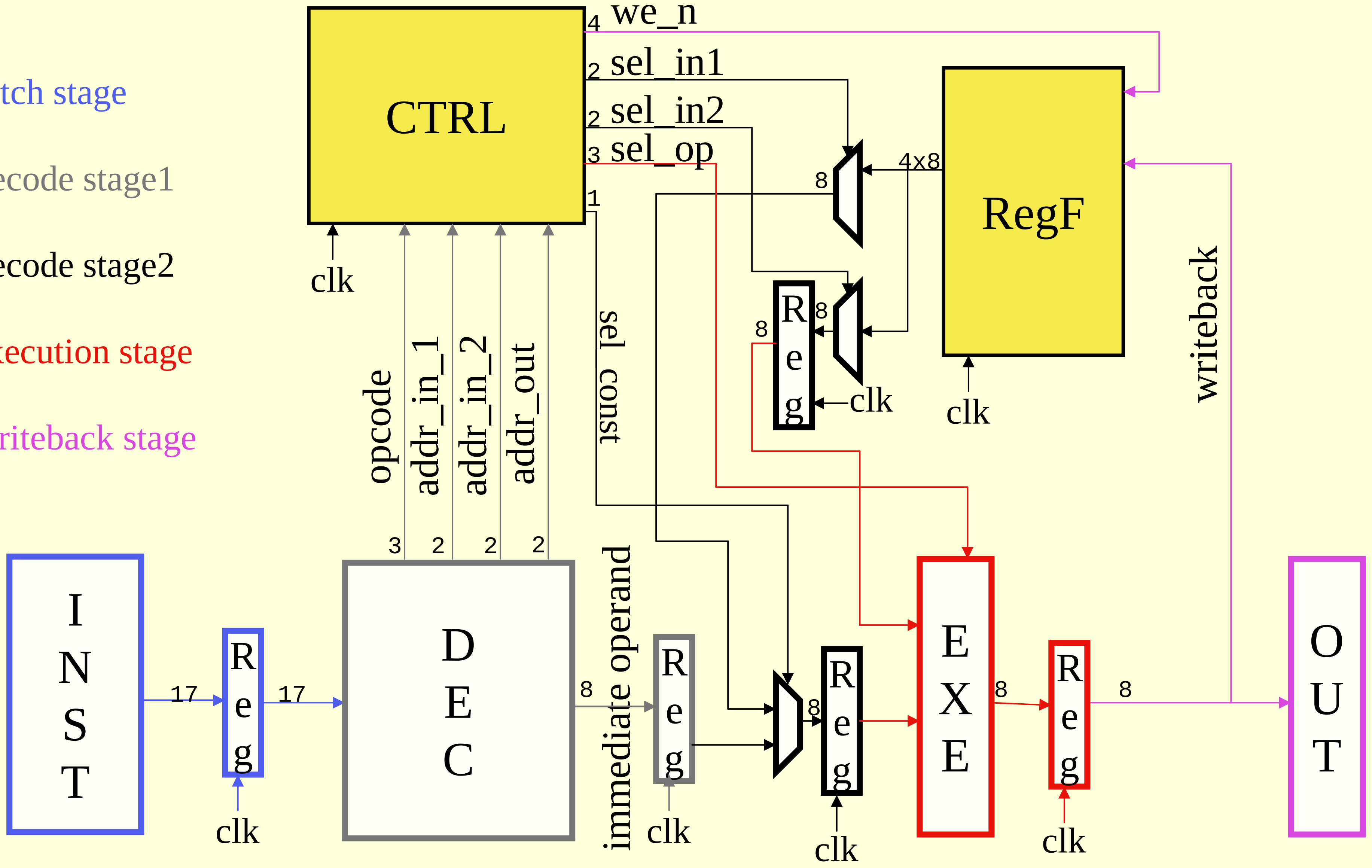
- Explanation of the processor architecture is attached with this document.
- A script to compile the program could be used in order to get the machine code. (Intranet)(Execute *./comp.py -h* to see how it works)
- A top level testbench with the functionality to read files with the program code in binary format will be used to test the processor. (Intranet)
- Submit your solutions to digiup@usi.ch at the latest on February 17

A simple microprocessor



How does it work?

- Fetch stage
- Decode stage1
- Decode stage2
- Execution stage
- Writeback stage



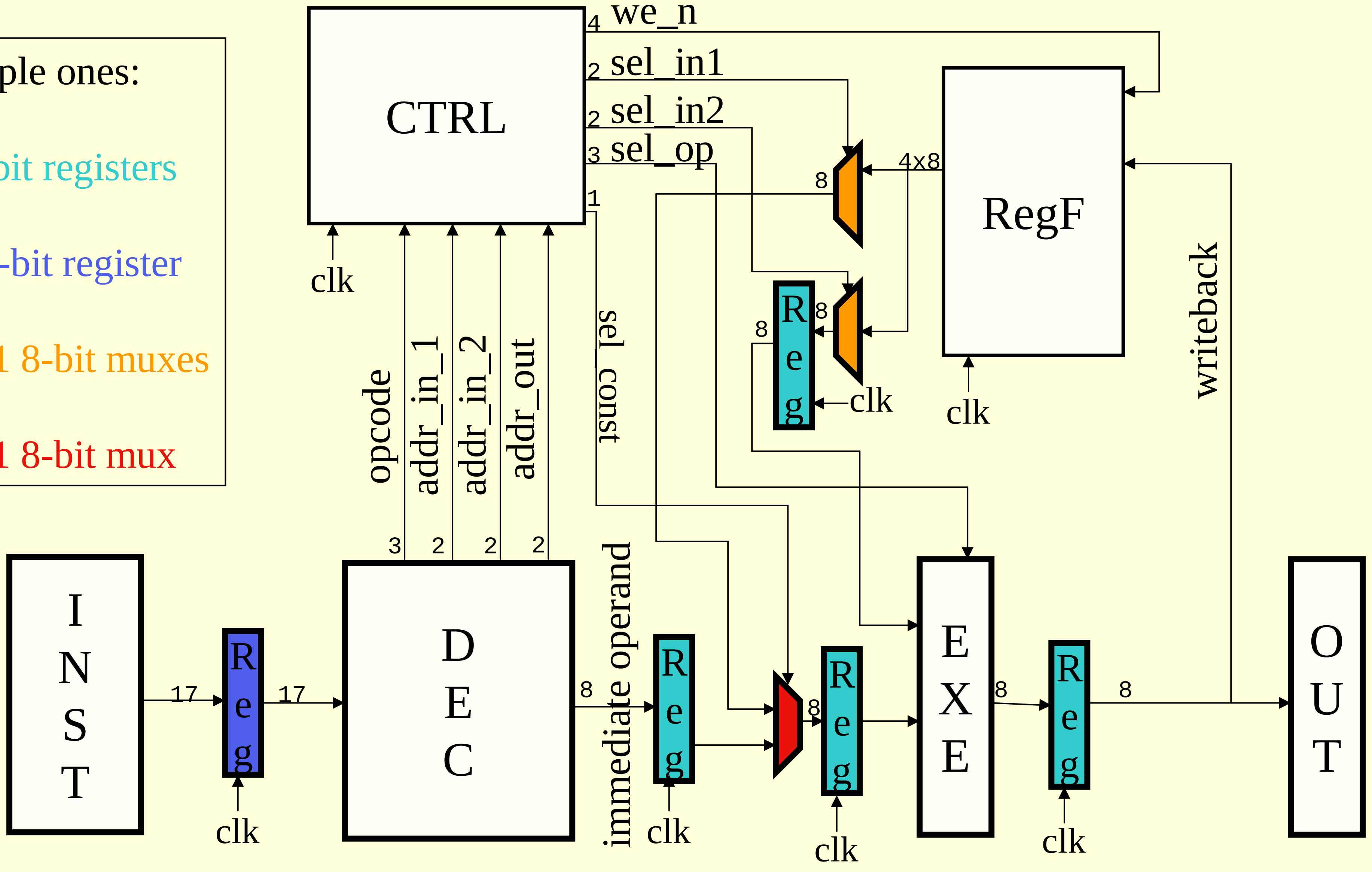
Processor description

- 17-bit instruction width, 8 bit data width
- 5-stage pipeline
- Sequential instruction flow only (no program counter, no jumps allowed)
- No forwarding
- No data cache
- 4-byte register file
- Instruction allowed:
  - MOVE_CONST (write a constant to the register file)
  - ADD
  - OR
  - AND
  - SL (shift left one position operand a)
  - NOP (do nothing this cycle)

Which component are needed?

Simple ones:

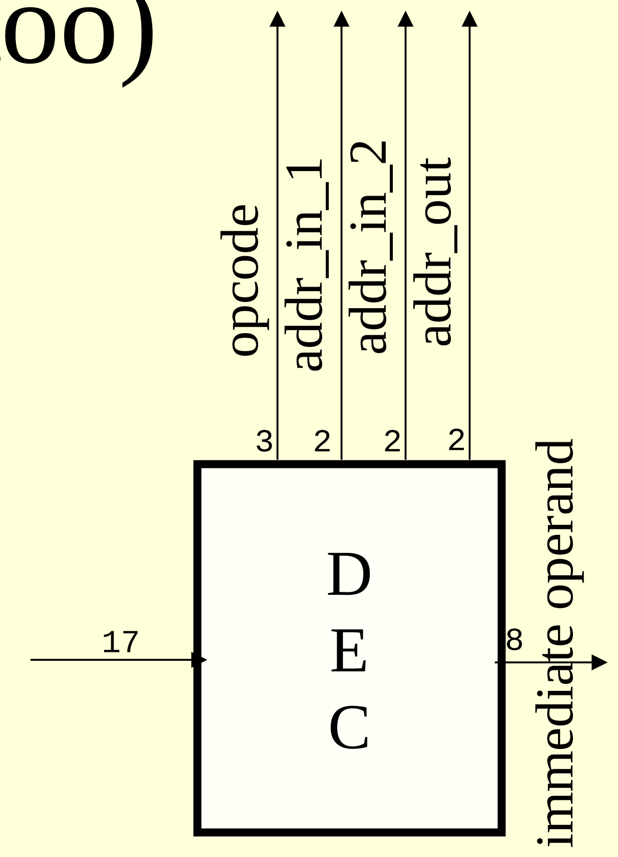
- 8-bit registers
- 17-bit register
- 4-1 8-bit muxes
- 2-1 8-bit mux

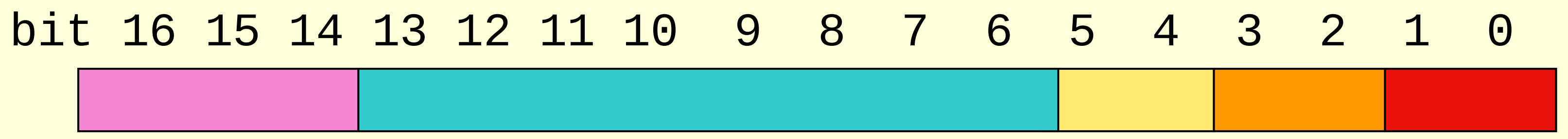


Decoder (easy too)

The decoder is a combinatorial circuit that takes the instruction as input and divides it into:

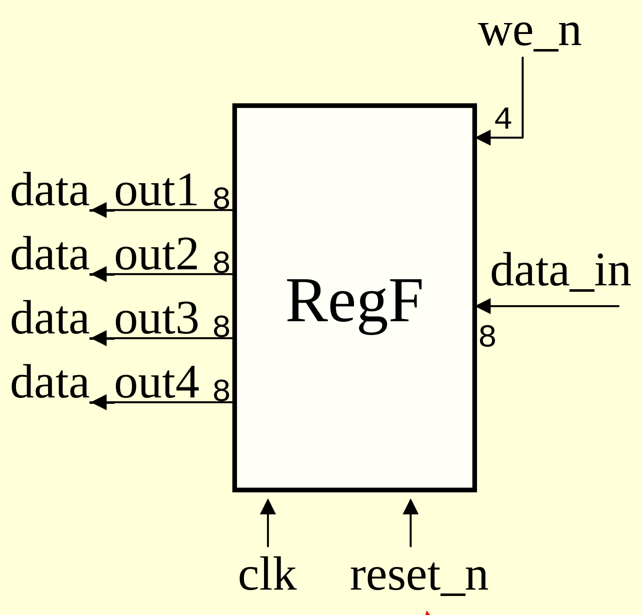
- opcode (bit 16 .. 14)
- immediate operand (bit 13..6)
- addr_in_1 (bit 5..4)
- addr_in_2 (bit 3..2)
- addr_out (bit 1..0)





Register file (medium easy)

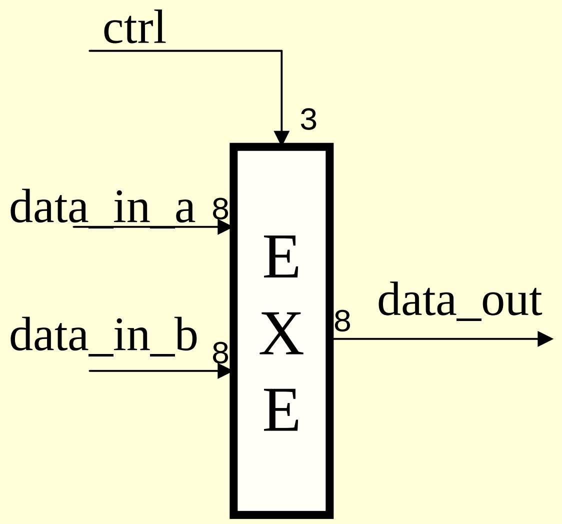
- The register file is a sequential memory circuit. In our microprocessor, the RF has four register, each of width 8.
- Each register accepts a new value when is write enable is **LOW**
- The process to update the RF is as follows:



```
WR_REG_FILE:PROCESS(s_clk, s_reset_n, s_datain, s_we_n)
      BEGIN
      IF (rising_edge(s_clk)) THEN
                IF (s_reset_n = '0') THEN
                          <reset_values of data_out(n)>
                ELSE
                          IF (s_we_n(0) = '0') THEN
                                    s_dataout1 <= s_datain;
                          END IF;
                          <repeat for the other data_out, with the proper we_n(i)>
                END IF;
      END IF;
END PROCESS;
```

every sequential element has a clk AND a reset_n (I omitted the reset_n elsewhere not to complicate slides even further…)

Execution unit (medium easy)



- The execution unit is combinatorial circuit, that executes the operation determined by the ctrl signal on the operands.

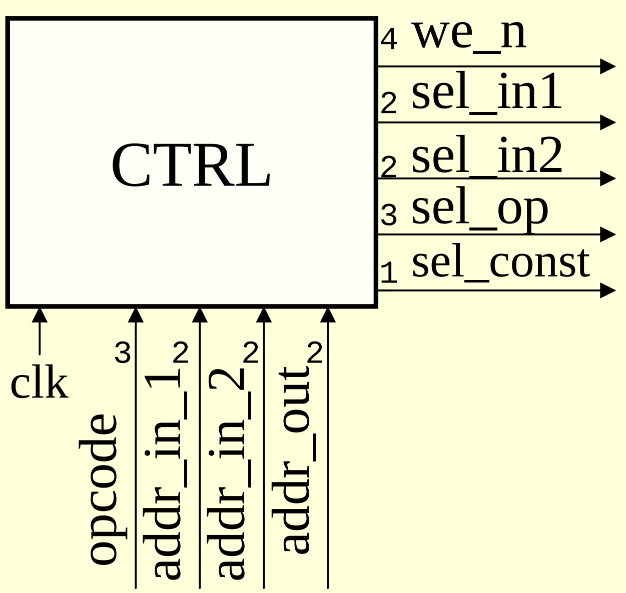
- The opcodes supported are:
  - "000" -> NOP (data_out <= (OTHERS => '0')
  - "001" -> MOVE_CONST (data_out <= data_in_a)
  - "010" -> ADD (data_out <= datain_a + datain_b)
  - "011" -> OR (data_out <= datain_a OR datain_b)
  - "100" -> AND (data_out <= datain_a AND datain_b)
  - "101" -> SL (data_out <= datain_a(6 DOWNTO 0) & '0')

Control unit (difficult)

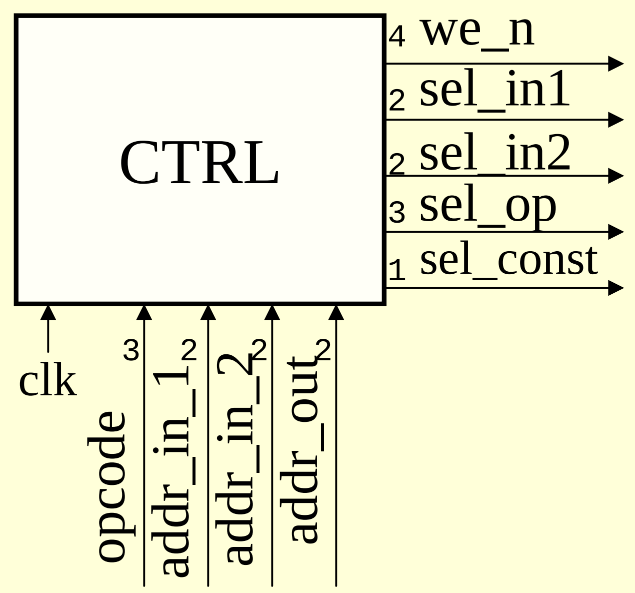
The control unit is a sequential circuit, governing control signals according to the (decoded) instruction **with the right timing**

- controls to the input multiplexer (sel_in_1, sel_in_2, sel_const) must be driven as soon as they are read
- controls to the execution unit must be delayed one clock cycle
- controls to the the RF (we_n) must be delayed two clock cycles; the encoding must be as follows:

```
IF (s_opcode = NOP) THEN
        s_we_mem2_n <= (OTHERS => '1');
ELSIF (s_outaddr = "00") THEN
        s_we_mem2_n <= "1110";
ELSIF (s_outaddr = "01") THEN
        s_we_mem2_n <= "1101";
ELSIF (s_outaddr = "10") THEN
        s_we_mem2_n <= "1011";
ELSE
        s_we_mem2_n <= "0111";
END IF;
```



Control unit (difficult)



Hints:

- to delay a signal one clock cycle:

```
PROC:PROCESS(s_clk, s_reset_n,…)
BEGIN
   IF (rising_edge(s_clk)) THEN
      IF(s_reset_n = ‘0’) THEN
         s_sig_mem <= ‘0’;
         s_sig_to <= ‘0’;
      ELSE
         s_sig_mem <= s_sig_from;
         s_sig_to <= s_sig_mem;    --s_sig_to equals s_sig_mem
      END IF;                      --delayed by one clock cycle
   END IF
END PROCESS;
```

- sel_const is ‘1’ when the operation is a MOVE_CONST operation
- we_n (and its delayed/anticipated versions) must be initialized to (OTHERS => ‘1’)
- we_n must be driven to (OTHERS => ‘1’) in a NOP operation (at the proper clock cycle), regardless of the output address

Putting all together (difficult)

- Connect all the previously defined elements in the way described on slide2
- _Find a good way to test the design.