Exercise 05: RISC-V - ALU, RF

Arithmetic Logic Unit (ALU)

The CPU consists of several blocks. One of these blocks is the Arithmetic Logic Unit (ALU) that allows to perform arithmetic and bitwise operations on binary numbers.

Write a 32-bit ALU using combinatorial logic (unclocked) that can perform 10 types of operations on 2 input operands:

- `ADD` - addition
- `SUB` - subtraction
- `AND` - bitwise `AND`
- `OR` - bitwise `OR`
- `XOR` - bitwise `XOR`
- `SLL` - logical shift left
- `SRL` - logical shift right
- `SRA` - arithmetic shift right
- `SLT` - less than
- `SLTU` - less than (unsigned)

Define an ALU entity with the following interface:

```
entity riscv_alu is
port (
    -- input operands
    i_s1        : in    std_logic_vector(31 downto 0);
    i_s2        : in    std_logic_vector(31 downto 0);
    -- select function (operation)
    i_funct3    : in    std_logic_vector(2 downto 0);
    i_funct7    : in    std_logic_vector(6 downto 0);
    -- output result
    o_d         : out   std_logic_vector(31 downto 0)--;
);
end entity;
```

`i_s1` and `i_s2` are input operands. `o_d` is the result output. The operation can be selected with the input ports `i_funct3` and `i_funct7`:

	`i_funct3(2 downto 0)`	`i_funct7(6 downto 0)`
ADD	000	0000000
SUB	000	0100000
SLL	001	0000000
SLT	010	0000000
SLTU	011	0000000
XOR	100	0000000
SRL	101	0000000
SRA	101	0100000
OR	110	0000000
AND	111	0000000

Note that `i_fucnt7` has only two valid values (`0000000` and `0100000`).

The SLT and SLTU instructions treat input operands as signed and unsigned respectively, and produce 1 if `i_s1` is less than `i_s2`.

The SLL, SRL and SRA instructions use `i_s1` as the value to be shifted and the shift amount is taken from lower 5 bits of `i_s2`. For SRA (arithmetic shift) the sign bit is copied into the vacated upper bits.

For more information on the RISC-V processor and supported arithmetic operations see https://github.com/riscv/riscv-isa-manual.

Instructions:

- Use either `when/else` chain or `if/else` in the combinatorial process to implement the logic of the ALU.
- Use `shift_left` and `shift_right` to implement shifts (covert inputs to signed and unsigned when needed).
- The default output of the ALU should be zero.

Register files (RF)

The register file is a collection of general-purpose registers that can be used to store intermediate values during execution of the CPU.

The base instruction set defines 32 registers:

- register `x0` that is hardwired to the constant 0 (i.e. writes are ignored and reads return 0)
- 31 general-purpose registers `x1-x31`

Implement a register file with following interface:

```
entity riscv_register_file is
port (
    -- read port 1
    i_raddr1  :   in  std_logic_vector(4 downto 0);
    o_rdata1  :   out std_logic_vector(31 downto 0);

    -- read port 2
    i_raddr2  :   in  std_logic_vector(4 downto 0);
    o_rdata2  :   out std_logic_vector(31 downto 0);

    -- write port
    i_waddr   :   in  std_logic_vector(4 downto 0);
    i_wdata   :   in  std_logic_vector(31 downto 0);
    -- write enable
    i_we      :   in  std_logic;

    i_clk     :   in  std_logic--;
);
end entity;
```

The register file has 3 ports: 2 read and 1 write.

Each read port is used to read the value of the addressed register and provide output on the same clock cycle (see wave diagram).

The write port updates the value of the register on the rising edge of the clock.

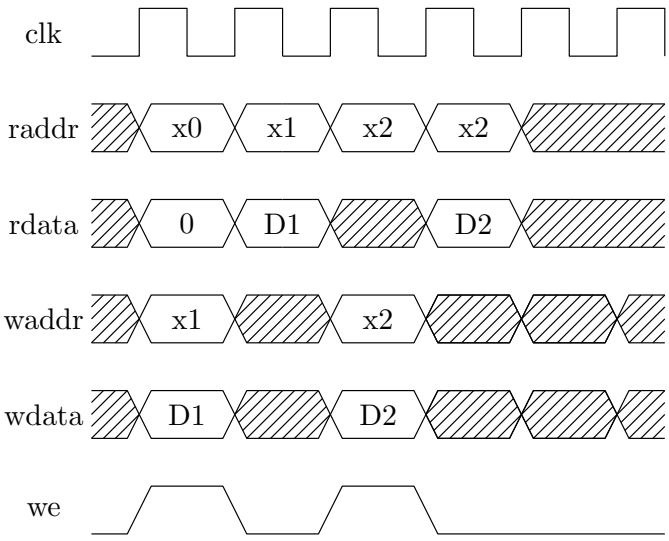

Figure 1: riscv_register_file

Instructions:

- define array type of 32-bit registers and corresponding signal with 32 registers
- define process that writes (`i_we = 1`) to selected (`i_waddr` address) register
- route register values to `o_rdata1`/`o_rdata2` ports according to addresses `i_raddr2`/`i_raddr2`
- handle register zero:
  - ignore write to register zero in write process
  - and implement selection logic that routes zero to read port when read address is zero

---

The labs take place in Staudingerweg 9 building: group 1 in room 4-516 on Thurday and group 2 in Newton room on Friday.

Send your solutions through MOODLE (https://moodle.uni-mainz.de).

The solution should include:

- Source code (only **your** `.vhd` files)
- Brief description if applicable, questions, etc.
- Screen shot of the simulation (use simulation range of 1 microsecond)
- Submit your files in one `.zip` archive
