Exercise 06: RISC-V - ID and RAM

Instruction decoder (ID)

The next block of the CPU is an instruction decoder. Instructions have a different format depending on which units (ALU, RAM, etc.) are in use. In case of RISC-V, the instructions can contain an opcode, functions, source/destination registers and an immediate (constant) value.

The RISC-V base integer instruction set (RV32I) contains 40 instructions. In this exercise we are going to implement an instruction decoder for the base instruction set that excludes byte and half word load/store instructions, and fence and system instructions (ecall and ebreak). This reduces the total to 31 instructions:

- 10 register-register (R-type: arithmetic),
- 11 register-immediate (I-type: arithmetic, indirect jump and load word),
- 2 upper immediate instructions (U-type: LUI and AUIPC),
- unconditional jump (J-type),
- 6 conditional branches (B-type),
- and store word (S-type) instructions.

A description of the instruction formats can be found in `riscv-spec-20191213-1.pdf` (ch. 2.3).

Instructions:

Implement an instruction decoder entity with the following interface:

```
entity riscv_instruction_decoder is
port (
    -- instruction opcode
    o_opcode    : out   std_logic_vector(6 downto 0);

    -- functions
    o_funct3    : out   std_logic_vector(2 downto 0);
    o_funct7    : out   std_logic_vector(6 downto 0);

    -- source register addresses
    o_rs1       : out   std_logic_vector(4 downto 0);
    o_rs2       : out   std_logic_vector(4 downto 0);
    -- destination register address
    o_rd        : out   std_logic_vector(4 downto 0);

    -- immediate value
    o_imm       : out   std_logic_vector(31 downto 0);

    -- input 32-bit instruction
    i_inst      : in    std_logic_vector(31 downto 0)--;
);
end entity;
```

Implement logic that decodes instruction input into opcode, functions, source and destination registers addresses and immediate value according to type of instructions.

See "RV32I Base Instruction Set" table in `riscv-spec-20191213-1.pdf` (page 130) for the detailed encoding format for each instruction type (including the encoding of opcodes and funct3/funct7 values). Use 7-bit opcode values from package `rv32i_pkg` (in `rv32i_pkg.vhd`) to identify instruction type.

The immediate value has to be assembled from bits of instruction according to table in Figure~2.4 of `riscv-spec-20191213-1.pdf`.

Note that if an instruction does not provide a value for a given output port, it should be set to zero (i.e. the default values for all outputs are zero).

Test your implementation with the provided testbench.

Random Access Memory (RAM)

Implement a Random Access Memory (RAM) with a read/write port that will be used by the CPU to load/store data.

The RAM should have the following interface:

```vhdl
entity riscv_ram is
port (
    -- address (1024 bytes = 256 words)
    i_addr      : in    std_logic_vector(9 downto 0);

    -- r/w data lines
    o_rdata     : out   std_logic_vector(31 downto 0);
    i_wdata     : in    std_logic_vector(31 downto 0);

    -- write enable
    i_we        : in    std_logic;

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

The RAM has word sized (32-bit) read/write ports and is byte addressable (`i_addr` specifies offset in bytes). However only word (4-byte) aligned read/write operations should be implemented and access to unaligned addresses should be ignored.

The latency for both write and read operations should be 1 clock cycle (after setting address the read data should be available on rising clock edge).

Test your implementation with the provided testbench.

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