Exercise 08: RISC-V CPU - part 1

R-type, I-type and LUI instructions

In this exercise we are going to implement a RISC-V processor that supports a subset of instructions:

- R-type instructions,
- I-type instructions,
- U-type instructions: LUI (load upper immediate) and AUIPC (add upper immediate to PC).

With this set, the CPU fetches instructions sequentially from Read-Only Memory (ROM), decodes the instructions and routes signals to the ALU and Register File (note that the above instructions read/write only from/to registers).

The LUI instruction is used to set the upper 20 bits of the destination register by storing the immediate value (IMM) into the register rd. Subsequently the lower bits can be set by using an I-type instruction by adding the immediate value to the previously set register.

The AUIPC is used to build pc-relative addresses by storing the value of PC + IMM in the rd register.

Instructions:

The CPU should have the following interface:

```
entity ex08_riscv_cpu is
port (
    -- ROM port (code)
    o_rom_raddr     : out   std_logic_vector(31 downto 0);
    i_rom_rdata     : in    std_logic_vector(31 downto 0);

    o_rf_wdata      : out   std_logic_vector(31 downto 0);

    -- active low reset
    i_reset_n   : in    std_logic;
    -- clock
    i_clk       : in    std_logic--;
);
end entity;
```

The `o_rom_raddr` is a port to access instruction memory (ROM) and the `i_rom_rdata` is the port that provides fetched instruction. As the ROM sets the `i_rom_rdata` port on the rising edge of the clock, the `o_rom_raddr` should be set to the next address of the program counter (PC) register. Introduce two signals: A `pc` register which stores the address of the current instruction and a `pc_next` signal that has the address of the next instruction. This CPU does not yet support flow control instructions (conditional and unconditional jumps) and in this exercise `pc_next` should have a value of `pc + 4` (the ROM is byte addressable, and instructions are 32-bit aligned). On each clock cycle the `pc` registers should be assigned to `pc_next`.

Write multiplexers that connect the ROM, instruction decoder, register file and ALU ports. The multiplexers should route signals depending on the value of opcode, e.g. the immediate output port of the instruction decoder should be connected to the write port of the register file for the LUI instruction and to the ALU output for I-type arithmetic instructions.

On reset (`i_reset_n = '0'`), the program counter register should be set to `0xFFFFFFFC`, such that the `pc_next` signal is zero and the instruction from address zero is fetched.

Test the CPU with the provided testbench and top file. The test program for the CPU is stored in the ROM file (`ex08_riscv_rom.vhd`) and correct execution is checked by examining output port `o_rf_wdata`. This port should track the value of (be connected to) the `wdata` port of the register file.

Try to decode what the program in the ROM is doing?

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
