Submission notes

- In this assignment you finish the project of this course.
- Have your VHDL files compressed and upload it with this format : **FinalProject_student_id_cpu.rar**
- Your explanation and report of the project part can be uploaded as a single PDF file with the following format
- **FinalProject_student_id_cpu.pdf**

Final Project: (100 points)

The goal of this experiment is to implement a simple version of processor we call it as 31L processor. The instruction format of this processor is as shown in figure1.

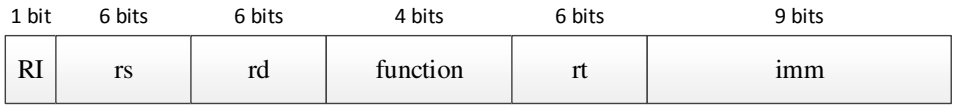

Figure 1. Instruction format

31L processor supports two types of instructions including R-type (RI='0') and I-type instructions (RI='1'). R-type is a register-based instruction and I-type is an immediate-based instruction. In R-type instructions the instruction is composed of three operands, two source registers, and one destination register. In immediate-based type, one of the sources is a 15 bit immediate value. Here is an example of R-type and I-type version of ADD instruction.

ADD RS, RD, RT // R_TYPE: rd ← rs + rt

ADDI RS, RD, IMM // I_TYPE: rd ← rs + immediate

In the R-type instructions rs and rt in the instruction specify the address of the source registers while rd contains the address of the output register and the last 9 bits of instruction (imm) are don't care. On the other hand, in the I-type instructions only one source register is needed and the value of the second input

This simple processor has a simple instruction set including the following commands:

Instruction	Description	Function code	Comments
NOP	nothing	"0000"	
ADD/ADDI	rd ← rs + rt / rd ← rs + immediate	"0001"	
SUB/SUBI	rd ← rs - rt / rd ← rs - immediate	"0010"	
COMP/COMPI	rs COMP rt / rs COMP immediate	"0011"	
SLT/SLTI	Set rd if rs < rt / Set rd if rs < immediate	"0100"	If the condition is satisfied set else reset destination
AND/ANDI	rd ← rs AND rt / rd ← rs AND immediate	"0101"	
OR/ORI	rd ← rs OR rt / rd ← rs OR immediate	"0110"	
NOT	rd ← NOT rs	"0111"	
XOR/XORI	rd ← rs XOR rt / rd ← rs XOR immediate	"1000"	
SLL/SLLI	rd ← shift left (rs) / rd ← shift left (rs)	"1001"	rt/ immediate value decides how many bits to shift
SRL/SRLI	rd ← shift right (rs) / rd ← shift right (rs)	"1010"	rt/ immediate value decides how many bits to shift
MOV/MOVI	rd ← rs / rd ← immediate	"1011"	

You are supposed to implement

- A controller/decoder which takes an instruction as input and then set the function types, register indexes, and all the other controlling signals in your design for choosing appropriate selector lines of the multiplexer.
- ALU component (composed of adder/subtractor, shifter, and comparator components). It has couple of inputs including two data bit lines and function type bit lines to implement the corresponding command. It also has 1 output data bit lines and 5 single bit lines including great, less, equal, carry, and overflow which will be activated in case they are needed.
- Register bank has four inputs (one clock and three register index inputs) and three outputs. The register index inputs contain the address of rs, rt and rd registers, the outputs contains the values of them.
- Implement the interconnection among all ALU, Register file, and Controller/decoder components. You can also employ multiplexers if they needed, in case of needing multiplexer, you need to implement a vhdl implementation of it and then employ an instance of it.
- A package including all the necessary constant values definition is provided for you which is called c31L_pack. You can add it to the beginning of any of your codes if needed.

We have provided the entity of three main component of this design as a hint:

Provided entities

Controller entity:

```
entity control is
  port(instruction       : in  std_logic_vector(BW-1 downto 0);
       rt_and_imm : : out  std_logic_vector (14 downto 0);
       read,write            : out  std_logic;
       rs_index     : out std_logic_vector(reg_field-1 downto 0);
       rt_index     : out std_logic_vector(reg_field-1 downto 0);
       rd_index     : out std_logic_vector(reg_field-1 downto 0);
       b_mux_sel    : out std_logic;
           alu_func     : out alu_function_type );
end;      --entity control
```

Register_bank entity:

```
entity reg_bank is
  port(clk            : in  std_logic;
       read,write            : in  std_logic;
       rs_index       : in  std_logic_vector(reg_field-1 downto 0);
       rt_index       : in  std_logic_vector(reg_field-1 downto 0);
       rd_index       : in  std_logic_vector(reg_field-1 downto 0);
       reg_source_out : out std_logic_vector(BW-1 downto 0);
       reg_target_out : out std_logic_vector(BW-1 downto 0);
       reg_dest_new   : in  std_logic_vector(BW-1 downto 0));

end; --entity reg_bank
```

ALU32 entity:

```
entity alu32 is
  port(a_alu32    : in  std_logic_vector(BW-1 downto 0);
       b_alu32    : in  std_logic_vector(BW-1 downto 0);
           alu_op  : in alu_function_type;
           g           : out std_logic;
           e           : out std_logic;
           l           : out std_logic;
           o_alu32: out std_logic_vector(BW-1 downto 0);
           c_alu32: inout std_logic;
           ov_alu32           : out std_logic);
end;
```

31L processor entity:

```
entity processor is
  port(clk          : in  std_logic;
       instruction      : in  std_logic_vector(BW-1 downto 0);
           great: out std_logic;
           less: out std_logic;
           equal: out std_logic;
           carry:out std_logic;
           over_flow:out std_logic);

end;
```

Notes:

- 31L processor does not support any load/store instructions for the sake of simplicity.
- It reads the operand from register file or as immediate values.
- The result of operation is supposed to be stored inside register file
- The immediate value has 15 bits while your whole design and registers are 32 bits, so you need a sign-extension component to modify your 15 bit immediate value into 32 bits. As a reminder a sign extension component increases the number of bits of a binary number while preserving the sign of number and its value. This is done by appending digits to the most significant location of the number.

You need to submit:

- The whole schematic of your design including all the components and their connection. You should use the same name as you have defined in your code.
- All the vhdl files needed for designing and implementing 31L processor
- Your testbench that is capable of verifying the accuracy of your design by providing all the instructions.