COMP4300

Lab Exercise Three

Objective

This lab develops the remaining datapath building blocks for the MIPS datapath. It will be combined with the MIPS control logic to make a working cpu in Lab 4.

Instructions

Develop VHDL for the following MIPS components. You should define an architecture for each of the entities given below. You should test each entity by developing simulation files for the entity. Your architecture should implement the functionality described in the text for each entity. To make the simulation results more readable, we will use a 32-bit datapath (both addr and data buses).

You should use the types from the dlx_types package you used in lab2, and the functions from bv_arithmetic as needed. The propagation delay through each unit should be 5 nanoseconds.

32-bit register . This will be used for the PC, NPC, A,B,Imm, ALUoutput, LMD, and IR registers. See Figure C.17 in textbook, as the RISC-V datapath has the same connectivity as the MIPS

```
entity dlx_register is
     port(in_val: in dlx_word; clock: in bit; out_val: out
dlx_word);
end entity dlx_register;
```

The register should be sensitive to all inputs. If `clock` is one, the value present at `in_val` should be copied to `out_val` . When `clock` goes to zero, the output value is frozen until `clock` goes high again.

32-bit Two-way multiplexer Selects one of two thirty-two bit inputs based on whether the `select` bit is one or zero. This has no state; it is purely combinational logic.

```
entity mux is
     port (input_1,input_0 : in dlx_word; which: in bit;
output: out dlx_word);
end entity mux;
```

Sign extender Takes a 16-bit input, produces a 32-bit output whose lower 16 bits are the same as the input, and whose upper 16 bits are copies of the (sign) bit from the input. This is combinational logic and has no internal state or variables.

```
entity sign_extend is
     port (input: in half_word; output: out dlx_word);
end entity sign_extend;
```

PC incrementer Adds 4 to its 32-bit input. Used to increment PC to create the value to put in NPC. An overflow is not indicated, you just wrap the address back around to zero (arithmetic mod 2**32). This is combinational logic and has no internal state.

```
entity add4 is
    port (input: in dlx_word; output: out dlx_word);
end entity add4;
```

Register File This is a dual-ported register file for reads, single ported for writes. It has a `clock` input that behaves just like the clock input on the single register defined above. Additionally it takes two 5-bit inputs regA and regB, each of which gives the address of the register whose 32-bit value is to be sent to the corresponding output port of the register file. There is a one-bit input Read_notwrite, that is set to a one do a read, a zero to do a write. Use the regA input as the number of the register to which the value in data_in is to be written for a write operation (read_notwrite = '0').

```
entity regfile is
     port (read_notwrite,clock : in bit;
           regA,rebB: in register_index;
         data_in: in dlx_word;
         dataA_out,dataB_out: out dlx_word
         );
end entity regfile;
```

Deliverables

Please turn in the following things for this lab:

- A printout of your VHDL code.
- Your simulation test file. Do not exhaustively test these designs since they take lots of input bits, but do test a reasonable number of things. For example, for the ALU, be sure to test every function, and for those that generate error codes, test the error conditions.
- Transcripts/screenshots of tests running your simulations. You cannot test exhaustively, but you should demonstrate that all your modules work.
-

Please turn in all files on Canvas. If I have questions, I may ask you to schedule a time to demo your code, if I can't figure out how something works by reading the code.