CSE4372/5392 (Spring 2025)
Lab #5

This lab is due by March 21, with a 10% penalty per week day for being late.

In this lab, you will construct the pipeline for the processor.

The following steps will guide you through this process:

1. Create a top level module that instantiates the dual_port_ram and rv32i_regs modules.

2. In your project top, use KEY0 (including the synchronizer from the seq_logic project in class) to provide the synchronous reset for the design. Use the 100 MHz clock (CLK100) as your system clock (we can use a slower clock later to make it easier to close timing). Terminate the inputs are necessary for the memory and register interfaces until they are needed later.

Using the cross-compiler, create the ram.hex file. The assembly language program should contain a series of R and I instructions separated by 3 NOPs between each pair of ALU instructions (no jumps or branches yet). Add a trailing EBREAK command immediately after the last instruction (no proceeding NOPs). The NOPs will allow the pipeline to operate without data hazard mitigation which will be addressed in the next lab.

3. Create a module, rv32_if_top, with the following port list:

```
// system clock and synchronous reset
input clk,
input reset,

// memory interface
output [31:2] memif_addr,
input [31:0] memif_data,

// to id
output reg [31:0] pc_out,
output [31:0] iw_out); // note this was registered in the memory already
```

4. Create a module, rv32_id_top, with the following port list:

```
// system clock and synchronous reset
input clk,
input reset,

// from if
input [31:0] pc_in,
input [31:0] iw_in,

// register interface
output [4:0] regif_rs1_reg,
output [4:0] regif_rs2_reg,
input [31:0] regif_rs1_data,
input [31:0] regif_rs2_data,

// to ex
output reg [31:0] pc_out,
output reg [31:0] iw_out,
output reg [4:0] wb_reg_out,
output reg wb_enable_out);
```

5. Incorporate your existing module, rv32_ex_top, with the following port list (note iw and pc outputs need to be added):

```
// system clock and synchronous reset
input clk,
input reset,

// from id
input [31:0] pc_in,
input [31:0] iw_in,
input [31:0] rs1_data_in,
input [31:0] rs2_data_in,
input [4:0] wb_reg_in,
input wb_enable_in,

// to mem
output reg [31:0] pc_out,
```

```
output reg [31:0] iw_out,
output reg [31:0] alu_out,
output reg [4:0] wb_reg_out,
output reg wb_enable_out);
```

6. Create a module, rv32_mem_top, with the following port list:

```
// system clock and synchronous reset
input clk,
input reset,

// from ex
input [31:0] pc_in,
input [4:0] wb_reg_in,
input wb_enable_in,

// to wb
output reg [31:0] pc_out,
output reg [31:0] iw_out,
output reg [31:0] alu_out,
output reg [4:0] wb_reg_out,
output reg wb_enable_out);
```

7. Create a module, rv32_wb_top, with the following port list:

```
// system clock and synchronous reset
input clk,
input reset,

// from mem
input [31:0] pc_in,
input [31:0] iw_in,
input [31:0] alu_in,
input [4:0] wb_reg_in,
input wb_enable_in,

// register interface
output regif_wb_enable,
output [4:0] regif_wb_reg,
```

output [31:0] regif_wb_data);

8. Instantiate rv32_if_top, rv32_id_top, rv32_ex_top, rv32_mem_top, and rv32_wb_top in the top level module.

9. In the rv32_if_top module, add a 32-bit register whose output will be the program counter (PC). The synchronous output of the register should set to a parameter PC_RESET if reset is active and PC+4 otherwise. This will allow the processor to execute a linear code flow (branching will be added in an upcoming lab). PC_RESET will be zero for our design. PC drives memif_addr directly. PC is registered to drive pc_out. The instruction word (memif_data) is registered in the memory module, so it directly drives iw_out. In the top level module of the project, connect the memory interface in IF to the memory module.

10. Register the input (pc_in and iw_in) to drive the outputs (pc_out and iw_out) in the ID, EX, and MEM stages of the pipeline.

11. By adding taps, you should now be able to see the PC and instruction word propagating through the entire pipeline. The contents of PC and the instruction word should be aligned and match the values in the hex file and assembly dump.

12. In the rv32_id_top module, decode iw to extract the rs1 and rs2 register numbers for the register interface. Register the rs1 and rs2 data values from the register interface for use in the EX stage. In the top level module of the project, connect the register interface in ID to the register module.

13. In the rv32_id_top module, decode iw to drive the writeback enable and writeback register number outputs. These should also be forwarded by the EX and MEM stages to the WB stage. In the WB stage, these signals should drive the register interface. The value of ALU registered from the EX stage should be routed to the writeback data value in the register interface. In the top level module of the project, connect the register interface in WB to the register module. The register value will be written back to the register on the clock edge the after data enters the WB stage.

14. By adding taps for the registers used in your assembly program, you should be able to see the ALU instructions properly reading and writing the registers.

15. Add logic to detect an EBREAK command. After detection, the pipeline should halt execution of instructions after the EBREAK instruction. The previous commands should all complete as normal.

16. Demonstrate operation of the instructions through the pipeline, showing the register operations in step 14 to the TA and the halting of the processor in step 15. Send your Verilog source .v or .sv files (don't send the entire project) to the TA for credit.