Pipelined Processor

Authors - Yehowshua Immanuel, Saloni Oswal, Tushar Krishna

Feb 8, 2021

Required Tools for this lab

- Brew
- iverilog
- GTKwave or Scansion

Lab Assignment 2

80 points

!!! warning The pipeline implementation in this lab probably has a number of errors and is not perfect, but is presented nonetheless as is for teaching purposes. It is sufficient for the seven or so supported instructions.

First click here to download the lab. Change into the `rtl` directory and run `make` to generate the `sim.vcd`, and then open it.

In this lab, you will be implementing the following:

1. *(Task 1: 20 points).* Add support for the load-to-use stall. For example, if a `lw` instruction produces data used by an immediately following instruction, one stall cycle should be included between the `lw` and the immediately following instruction.

Currently the processor does not stall when the instruction following the load depends on it. The logic that needs to be repaired to add load-to-use stall is in the file `STALL_CONT.sv`. You can use op_stall signal.

```
    assign RS_EX_hazard  = 0;
    assign RS_MEM_hazard = 0;
    assign RS_WB_hazard  = 0;

    //Check to see if any of the stages have RS hazards
    logic RS_hazard;
    assign RS_hazard = RS_EX_hazard | RS_MEM_hazard | RS_WB_hazard;

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Check if there is a hazard on RT
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    //(rt(IRID)==destEX)  && use_rt(IRID) && RegWriteEX     or
    //(rt(IRID)==destMEM) && use_rt(IRID) && RegWriteMEM    or
    //(rt(IRID)==destWB)  && use_rt(IRID) && RegWriteWB
    logic RT_EX_hazard  ;
    logic RT_MEM_hazard ;
    logic RT_WB_hazard  ;
    assign RT_EX_hazard  = 0;
```

```
assign RT_MEM_hazard = 0;
assign RT_WB_hazard  = 0;
```

For example, `RT_EX_hazard` is not always `0`, it is sometimes `1`. To give you a hint, `RT_EX_hazard` depends on `ip_dest_EX`, `use_RT`, and `ip_RegWrite_EX`.

2. *(Task 2: 40 points).* Implement data forwarding to the EX stage. There is a blank module called `FWD_CONT.sv` already created for you and instantiated within MIPS.sv. Feel free to modify the input/output ports if required in your implementation. Just how you checked for hazards in the `STALL_CONT.sv` file, you will check for hazards and forward the correct value from EX/MEM or MEM/WB stage to the execute unit.
3. *(Task 3: 20 points).* Implement flushing in the pipeline. Assume branches are predicted as not taken. Thus, the fetch stage will keep fetching instructions through PC+4 until the branch condition is resolved. Note that if the branch is taken, then the instructions which were speculatively fetched would have to be flushed. Flushing out will mean clearing out the relevant (before branch resolution) registers.

It might help to start by filling out the table below. The number at the top of the table represent the cycle count. You can access the cycle count by viewing the `cycle_cnt` signal in `MIPS_pipelined_tb` in the waveform viewer.

You'll notice that the first instruction in the table has already been filled in for you. Also, if you open `IFETCH.sv`, you'll notice that the instructions in the table below match those in the `inst_ram`

	3	4	5	6	7	8	9	10	11	12	13	14	15	16
lw $2, 0	F	D	E	M	WB									
lw $3, 0x4														
add $1, $2, $3														

You can use a combination of lw, add, sub, beq to test out stalling, forwarding, and flushing.

Submission

Submit a zip file of the rtl directory with your changes. Make sure you add comments for the changes.