# Introduction to Digital Logic Design Lab EECS 31L

Lab 4:

Pipeline processor

2/25/2021

## 1 Objective

In this lab, I designed a pipeline processor. By using a pipeline processor, compared to a single cycle processor, we can get a better performance by pipelining instructions.

## 2 Procedure

**Implementing**

One of the most different part between a pipeline and a sigle cycle processor is that pipleline processor has 4 registers called IF/ID, ID/EX, EX\_MEM, MEM\_WB.

There were 5 parts that I revised. 1, IF\_pipe\_stage. 2, ID\_pipe\_stage. 3, EX\_pipe\_stage. 4, EX\_forwarding\_unit. 5, hazard\_detection. 6, mips\_32.

For IF\_pipe\_stage, I added 2 mux that used for branch (sel: branch\_taken) and jump (sel: jump), and defined pc\_reg. I calculated pc\_plus 4 by adding 4 and pc\_reg. The other output instr is generated by instruction\_mem. I connected them by pipe\_reg\_en.

For IF/ID REG, I connected output from IF (pc\_plus4, instr) and other 4 inputs (clk, reset, en(data\_hazard), flush(IF\_Flush) )as the input, and created 2 outputs (if\_id\_pc\_plus4, if\_id\_instr).

For ID\_pipe stage, there were 8 inputs (clk,reset, if\_id\_instr, pc\_plus4, mem\_wb\_reg\_write, mem\_wb\_write\_back\_data, mem\_wb\_write\_reg\_addr, Data\_Hazard, Control\_Hazard) and 13 outputs (reg1, reg2,mm\_value,branch\_address,jump\_address,branch\_taken,destination\_reg,mem\_to\_reg,alu\_op,mem\_read, mem\_write,alu\_src,reg\_write,jump) I connected the input and output based on the description of lab manual. There were 4 modules inside ID\_pipe\_stage: reg, control, sign-extend, and control, and there were all related to instr. Control was connected to instr[31:26], its ouput mem\_to\_reg, mem\_read, mem\_write, alu\_scr, reg\_write were control by a mux connected to (~(Data\_Hazard) | Control\_Hazard), and jump is a direct ouput, and Branch was connected to a AND gate. Instr[25:0] was shifted by 2 and directly generated a ouput called jump\_address. Pc\_plus 4 was added with sign-exteneded instr[15:0] shifted left 2 and their sum was a output called branch\_address. Instr[25:21] ,instr[20:16], and mem\_wb\_reg\_write, mem\_wb\_write\_reg\_addr and mem\_wb\_write\_back\_data were connected to internal module Reg\_File as 5 inputs. The 2 outpus of Reg\_File were called reg1 and reg2, and also both were connected to a Eq\_test module and AND with a branch to generate a ouput called branch\_taken. The ouput imm\_value was sign-extened from instr[15:0] and destination\_reg was generated by mux2(a: instr[20:16], b:instr[15:11],sel:reg\_dst from control)

For ID/EX reg, there were 11 inputs and 11 outputs, I just directly connected them by pipe\_reg.

For Harzard\_Detection module, the code was provided so I copy/paste the code directly.

For execution module, there were 4 internal modules called mux2, mux4, alu\_control and alu. 8 inputs (id\_ex\_instr,reg1,reg2,id\_ex\_imm\_value,ex\_mem\_alu\_result,mem\_wb\_write\_back\_result,id\_ex\_alu\_src,id\_ex\_alu\_op,Forward\_A, Forward\_B) and 2 outputs (alu\_in2\_out, alu\_result) were there. Alu\_control’s 2 input were id\_ex\_instr[5:0] and id\_ex\_alu\_op, its output alu\_control was connected to alu. There were 2 mux4 Mux4 1(mem\_wb\_write\_back\_result), .c(ex\_mem\_alu\_result), .sel(Forward\_A), and its output went to the alu. Mux4 2 mem\_wb\_write\_back\_result), .c(ex\_mem\_alu\_result), .sel(Forward\_B) and its output went to the execution output called alu\_int\_out. Another mux2 (.a(mux2\_output), .b(id\_ex\_imm\_value),

.sel(id\_ex\_alu\_src)) was also there and its output was the other input of alu. Alu’s output was the one of the outputs of execution called alu\_result.

For forwarding unit, there were 6 inputs (ex\_mem\_reg\_write,ex\_mem\_write\_reg\_addr,id\_ex\_instr\_rs,

id\_ex\_instr\_rt,mem\_wb\_reg\_write,mem\_wb\_write\_reg\_addr)and 2 outputs(Forward\_A, forward\_B). I then directly translated the logic in the lab manual to Verilog code.

For EX.MEM REG, there were 8 inputs and 8 outputs and I directly connected them by pipe\_reg.

For Memory (Data\_memory), there were 5 inputs (clk,mem\_access\_addr,mem\_write\_data,

mem\_write\_en,mem\_read\_en, )and 1 output(mem\_read\_data). I connected them with the module data\_memory directly with those wires.

For MEM/WB REG, there were 5 inputs and 5 outputs, and I directly connected them with pipe\_reg.

For Write back. There were 3 inputs and 1 output (.a(mem\_wb\_alu\_result),

.b(mem\_wb\_mem\_read\_data), .sel(mem\_wb\_mem\_to\_reg), .y(write\_back\_data), and I connected them by a Mux. The ouput of this mux also was the ouput of the processor.

**Debugging**

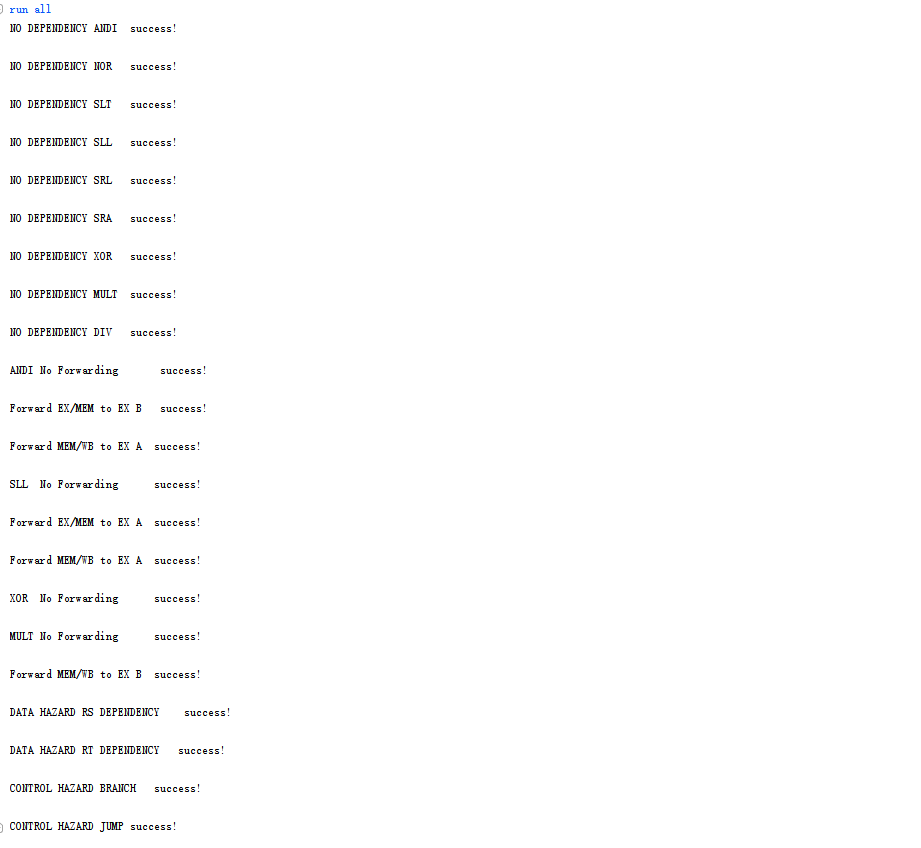
One of the most challenging things to do in this lab was to debug the code because this project was complicated, and a lot of wires were connected. It took me 4 hours total to debug this stuff.

When I just finished writing my code, I was not able to get it simulated because there were several errors in my code. I forgot to name each module I used in mips\_32. After I got all errors fixed, I was able to run it.

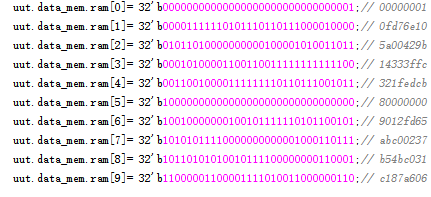
I only got 3 success at the first time. Firstly, I checked console at the bottom of the screen and found a lot of warming regarding the bits of mux. I realized that I defined the width of some mux in a wrong way. After I fixed that and had zero warming, I still got 3 success.

I was stuck at that time and I went to my mips\_32 code, 3 hours later, I found I swapped the definition of 2 of the wires. Lastly, I was able to get all success.

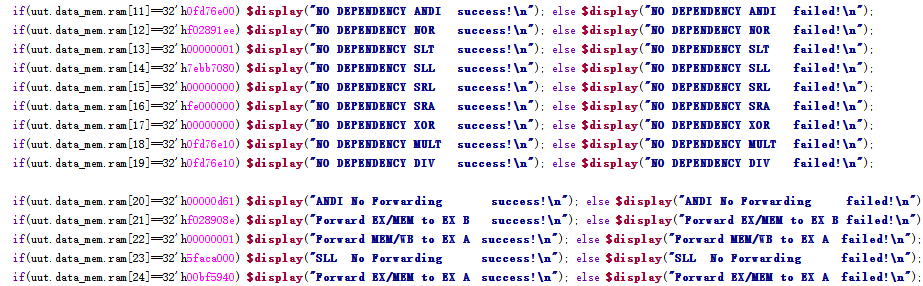
## 3 Simulation Results

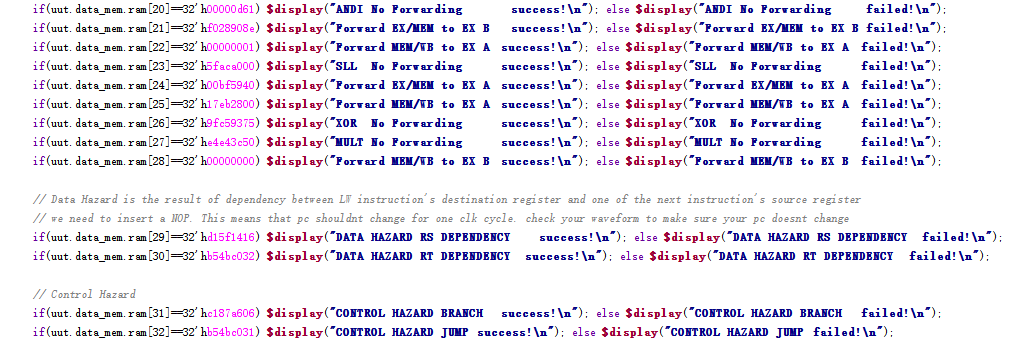


**According to the Testbench, we firstly put some data for the processor.**

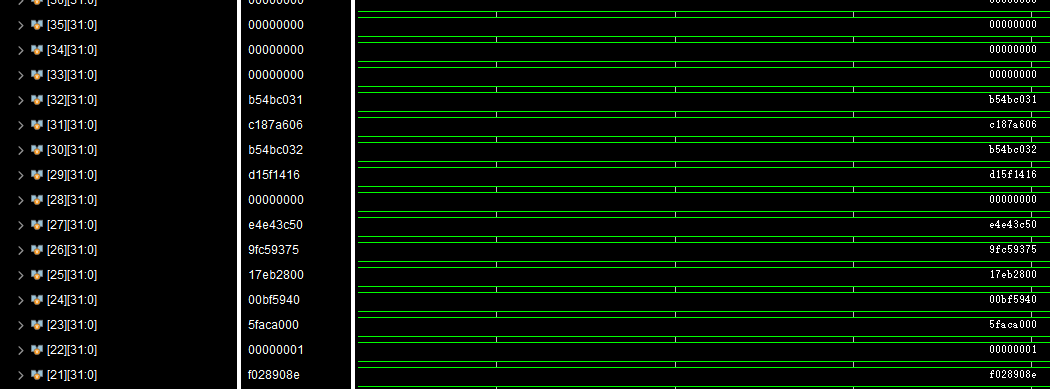


**Then, if all the tests were passed, there will have the following results in the data memory.**

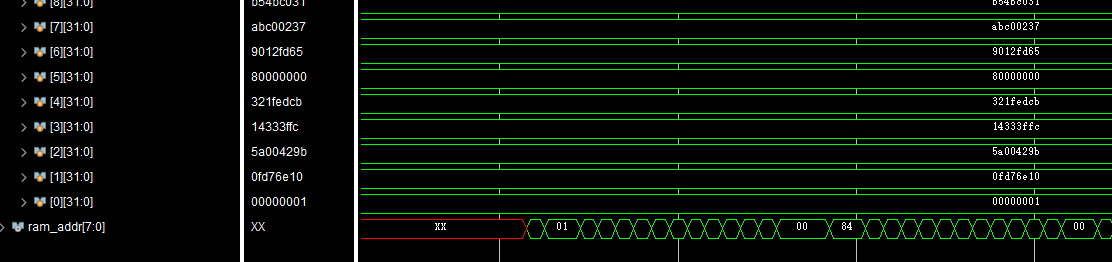




**Then, go back to the wave form of my data memory.**







**By comparing the actually results displayed in the waveform and the expected results, all results were verified and therefore the simulation was success.**

