CECS 530 - Lab 8

"Making LEGv8 Pipeline"

Due date: December 10, 2020

Student Name: Julie Kim

[redacted]

I certify that this submission is my original work

Julie Kim—December 10, 2020

Your signature or electronic signature

Lab Report: Lab Assignment 8- "Pipeline"

1. **Goal:**
   To build a pipeline process. By using all the units built from previous labs including Instruction Memory, Register file, Alu and Data Memory. All the signals from Instruction memory are sent stage by stage to the Data Memory and back to Registers File. By inserting Pipeline Register between each units, we would like to improve the execution time from 16 clock cycles to less when executing 16 instructions. When the pipeline is filled up after 4 clock cycles, there will be instruction output every clock cycle.

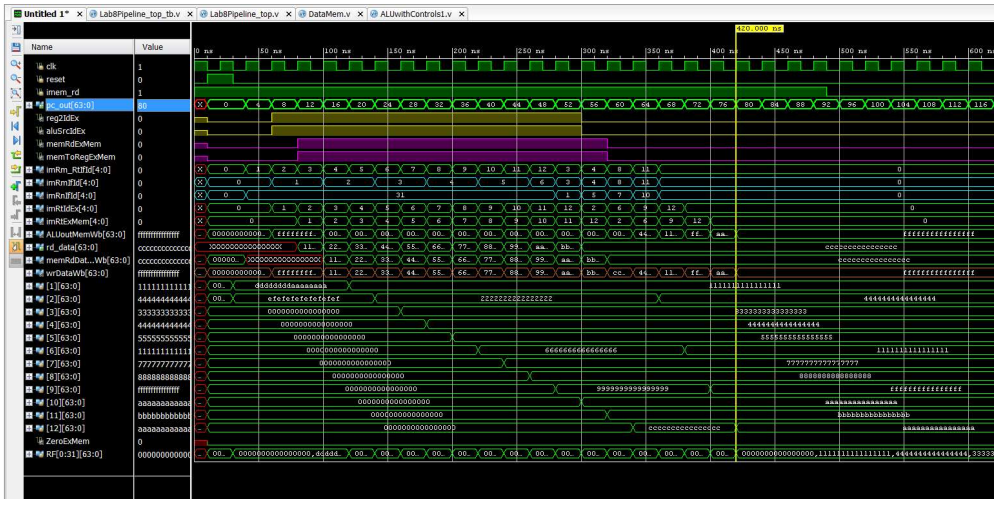
2. **Steps:**
   - Separate and connect each unit with signals. One Instruction unit, connected to one Registers File, connected to one Alu, connected to one Memory. All the 5 units are connected via internal signals and are in one Top module.
   - Build 4 muxes, for PC, Register file, ALU, and last one for the memory instruction
   - Build 4 pipeline register to hold the signals passing from Instruction memory all the way to Data Memory and back to Register file.
   - The first pipeline register called IF/ID holds signals as 64-bit PC, 32-bit. Total of 96-bit
   - The second pipeline register named ID/EX hold signals as 9-bit decoder output, 64-bit PC, 64-bit read data 1 or 2, 64-bit Signed extended output, 11-bit opcode, and 5-bit instruction for ALU result and load from memory to register. Total of 281 bits
   - The third pipeline register named EX/MEM hold signals as 1-bit RegWrite, 1-bit MemtoReg, 1-bit Branch, 1-bit MemRead, 1-bit MemWrite, 64-bit of branch adder, 1-bit Zero, 64-bit of ALU result, 64-bit of memory data and 5-bit of write back address. Total of 203 bits.

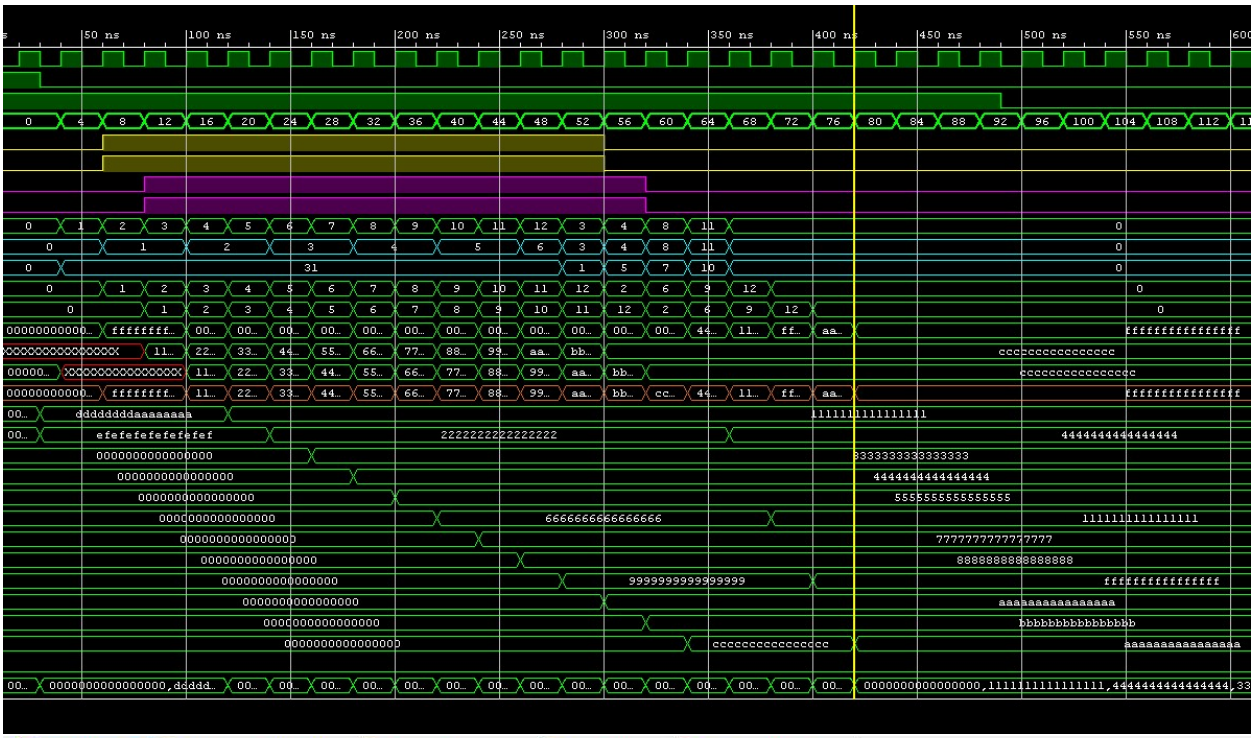
- The fourth pipeline register named MEM/WB holds signals as 1-bit RegWrit, 1-bit MemtoReg, 64-bit Read data, and 64-bit of ALU result and 5-bit write back address

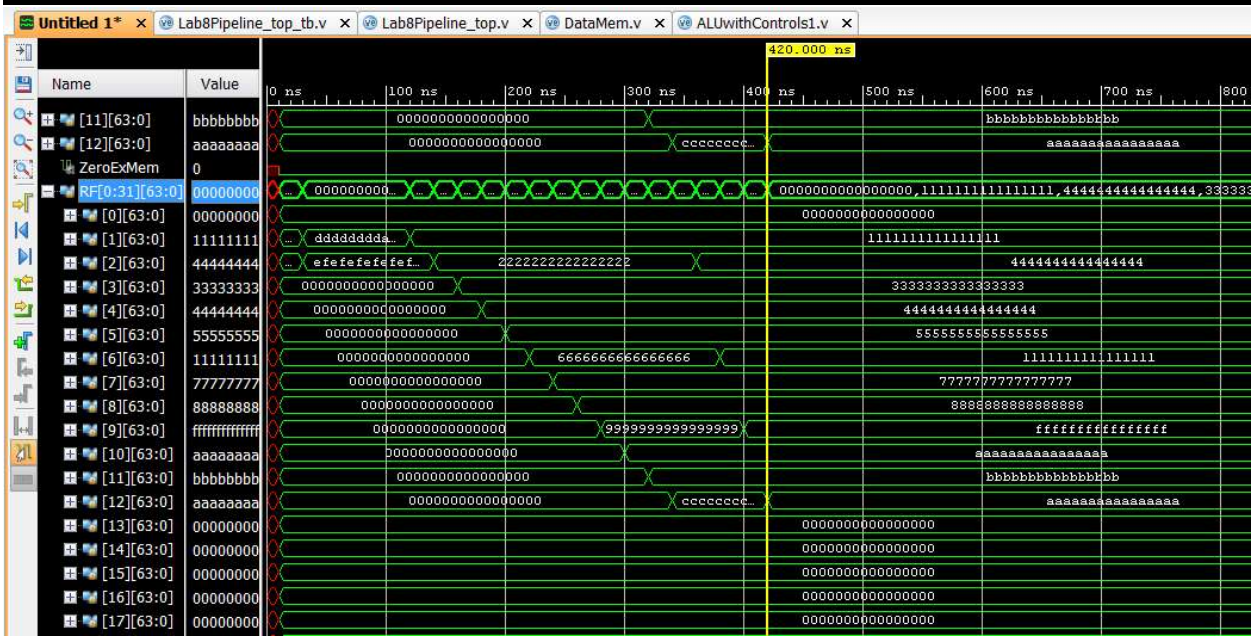
3. **Results:**
   - The Load instruction fetch data from memory 64'b 0 at byte 0 through 64'b ccc… of byte 96. Write the result in register address R1 through R12 with the result of 64'b111… through 64'bccc.
   - Look at register file, R1 is 64'h11.., R2 is 64'h222…, R3 is 64'h333,……. R12 is 64'hccc….
   - Content in R1, R3, R5, R4, R7, R8, R10, and R11 are used for the last 4 arithmetic operations and store back in R2, R6, R9, and R12 with a different results
   - The program executes 16 instructions, 12 load and 5 arithmetic instruction in total of 20 clock cycles.
   - The PC increment by PC+4 until it fetches all the 16 instructions
   - After 4 clock cycle the pipeline is filled up, and at the edge of 5th clock, the first output is ready
   - After 4 clock cycles, one instruction is finished every clock cycle.
   - The speed up is 16/(20*(1/5)) = 4
4. **Conclusion:**

From this lab I learn how to construct pipeline register and connect the signals needed from each stage to the next. The challenge encounter is the confusion and missing signals resulted in no result in the ALU operation or read data from memory. The connection of result needed to be concrete to get the right simulation result. Debugging process is a challenge because there is no clue from the IDE. Visually and manually tracing each signal is a lot of work. By trying to study the signal connection very carefully before writing the code helped me to get this lab done on time.

Simulation Results






Untitled 1*
Lab8Pipeline_top_tb.v
Lab8Pipeline_top.v
DataMem.v
ALUwithControls1.v
Name
Value
420.000 ns
[11][63:0]
[12][63:0]
ZeroExMem
RF[0:31][63:0]
[0][63:0]
[1][63:0]
[2][63:0]
[3][63:0]
[4][63:0]
[5][63:0]
[6][63:0]
[7][63:0]
[8][63:0]
[9][63:0]
[10][63:0]
[11][63:0]
[12][63:0]
[13][63:0]
[14][63:0]
[15][63:0]
[16][63:0]
[17][63:0]

```
Register_0=0000000000000000
Register_1=1111111111111111
Register_2=4444444444444444
Register_3=3333333333333333
Register_4=4444444444444444
Register_5=5555555555555555
Register_6=1111111111111111
Register_7=7777777777777777
Register_8=8888888888888888
Register_9=ffffffffffffffff
Register_10=aaaaaaaaaaaaaaaa
Register_11=bbbbbbbbbbbbbbbb
Register_12=aaaaaaaaaaaaaaaa

Data Memory
DM[8]=1111111111111111
DM[16]=2222222222222222
DM[24]=3333333333333333
DM[32]=4444444444444444
DM[40]=5555555555555555
DM[48]=6666666666666666
DM[56]=7777777777777777
DM[64]=8888888888888888
DM[72]=9999999999999999
DM[80]=aaaaaaaaaaaaaaaa
DM[88]=bbbbbbbbbbbbbbbb
DM[96]=cccccccccccccccc
relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:12 . Memory (
```

Coding:

```
module Lab8Pipeline_top(clk, reset, imem_rd);

    input clk, reset, imem_rd;

    //internal signal
    wire [31:0]  imem_out_wire;
    wire [63:0]  pc_out_wire;
    wire [63:0]  branch_addr_wire;
    wire         Zero_wire , Branch_wire ;
    //IFstg_1================================================
    InstructionMem1  INSTRUCTION_MEM_unit(
                     .clk(clk),
                     .reset(reset),
                     .imem_rd(imem_rd),
                     .imem_out(imem_out_wire),
                     .pc_out(pc_out_wire),
                     .is_branch(Branch_wire),
                     .branch_addr(branch_addr_wire),
                     .Zero(Zero_wire));

    //IF/ID pipeline register*************
    wire [299:0] ifIdOut_wire;
    wire    [10:0] opCfieldIfId;
    wire    [31:0] imem_if_id;
    wire    [63:0] pc_if_id;
    IF_ID_reg   IF_ID_PR_unit(
                     .clk(clk),
                     .reset(reset),
                     .instr(imem_out_wire),
                     .pc(pc_out_wire),
                     .if_id_out(ifIdOut_wire));

    assign  opCfieldIfId = ifIdOut_wire[31:21];
    assign  imem_if_id = ifIdOut_wire[31:0];
    assign  pc_if_id   = ifIdOut_wire[127:64];
```

```
//IDstg_2================================================
wire Reg2Loc_wire, ALUSrc_wire, MemtoReg_wire, MemRead_wire;
wire RegWrite_wire, MemWrite_wire, ALUOp1_wire, ALUOp0_wire;
wire [4:0]   imRnIfId, imRmIfId, imRm_RtIfId; // rd_addr_2_wire;
wire [18:0]  offset_addr;
wire [63:0]  rd_data_1_wire, rd_data_2_wire;
wire [63:0]  ALU_result_wire;
wire [63:0]  SE_offset_addr;
wire [63:0]  wrDataWb, mem_rd_data_wire;
wire         regWrMemWb, memToRegMemWb;
wire [4:0]   imRtMemWb;
InstructionDecoder  INSTRUCTION_ID_unit(
                    .Opcode(opCfieldIfId),
                    .Reg2Loc(Reg2Loc_wire),
                    .ALUSrc(ALUSrc_wire),
                    .MemtoReg(MemtoReg_wire),
                    .RegWrite(RegWrite_wire),
                    .MemRead(MemRead_wire),
                    .MemWrite(MemWrite_wire),
                    .Branch(Branch_wire),
                    .ALUOp1(ALUOp1_wire),
                    .ALUOp0(ALUOp0_wire));
registersFile REGFILE_unit(
                    .clk(clk),
                    .reset(reset),
                    .wr_en(regWrMemWb),
                    .wr_addr(imRtMemWb),
                    .wr_data(wrDataWb),
                    .rd_addr_1(imRnIfId),
                    .rd_addr_2(imRm_RtIfId),
                    .rd_data_1(rd_data_1_wire),
                    .rd_data_2(rd_data_2_wire));
```

```
                    _    _    _    _ _
90
91      //1. Reg2Loc==1, Rt[4:0]    //for laod address load Rt, offset[Rn]
92      //0. Reg2Log==0, Rm[20:16]  //for Add Rm       Add  Rd, Rn, Rm
93      assign imRm_RtIfId = (Reg2Loc_wire)? imem_if_id[4:0] : imem_if_id[20:16];
94                                          //Rt or Rd[4:0]   //      Rm[20:16]
95      assign imRnIfId       = imem_if_id[9:5];
96      assign imRmIfId       = imem_if_id[20:16];
97      //check if it is branch instruction
98      assign offset_addr = (Branch_wire)? imem_if_id[23:5] : {10'b0, imem_if_id[20:12]};
99      assign SE_offset_addr = {45'b0, offset_addr};
00
01      //ID_EX pipeline register
02      wire    [280:0]  idExIn_wire, idExOut_wire;  //281 bits
03      wire    reg2IdEx, aluSrcIdEx, aluOp1IdEx, aluOp0IdEx, regWrIdEx;//[280:277]
04      wire    memToRegIdEx, brIdEx, memRdIdEx, memWrIdEx;             //[276:272]
05      wire    [10:0]  opCFieldIdEx;                                   //[271:261] 11-bit
06      wire    [63:0]  pcIdEx, SE_AddrIdEx;                            //[260:197], [196:133]
07      wire    [4:0]   imRtIdEx;                                       //[132:128]
08      wire    [63:0]  rDataA1IdEx, rDataB2IdEx;                       //[127:64], [63:0]
09      wire    [63:0]  brAddrIdEx;
10      ID_EX_reg   ID_EX_PR_unit (
11              .clk(clk),
12              .reset(reset),
13              .idExIn({19'b0, idExIn_wire}),
14              .idExOut(idExOut_wire)
15              );
16      assign  idExIn_wire = {Reg2Loc_wire, ALUSrc_wire, ALUOp1_wire, ALUOp0_wire, //[280:277]
17                             RegWrite_wire,MemtoReg_wire,Branch_wire,MemRead_wire,MemWrite_wire,//[276:272]
18                             imem_if_id[31:21], pc_if_id, SE_offset_addr, //[271:261] 11-bit, [260:197], [196:133]
19                             imem_if_id[4:0], rd_data_1_wire, rd_data_2_wire };   //[132:128], [127:64], [63:0]
20
21      assign  reg2IdEx        = idExOut_wire[280];
22      assign  aluSrcIdEx      = idExOut_wire[279];
23      assign  aluOp1IdEx      = idExOut_wire[278];
24      assign  aluOp0IdEx      = idExOut_wire[277];
25      assign  regWrIdEx       = idExOut_wire[276];
26      assign  memToRegIdEx    = idExOut_wire[275];
27      assign  brIdEx          = idExOut_wire[274];
28      assign  memRdIdEx       = idExOut_wire[273];
29      assign  memWrIdEx       = idExOut_wire[272];
30      assign  opCFieldIdEx    = idExOut_wire[271:261]; //11 bits
31      assign  pcIdEx          = idExOut_wire[260:197]; //64 bits
32      assign  SE_AddrIdEx     = idExOut_wire[196:133]; //64 bits
33      assign  imRtIdEx        = idExOut_wire[132:128]; //5 bits
34      assign  rDataA1IdEx     = idExOut_wire[127:64];
35      assign  rDataB2IdEx     = idExOut_wire[63:0];
36
37      assign  branch_addr_wire = (brIdEx)? (SE_AddrIdEx << 2): 64'b0;
38      assign  brAddrIdEx = pcIdEx + branch_addr_wire;
39
```

```
assign  brAddrIdEx    = pcIdEx + branch_addr_wire;

//Exstg-3================================================================================
ALUwithControlsl ALUwithCONTROL_unit(
        .addr_cal(aluSrcIdEx),
        .ALU_SE_addr(SE_AddrIdEx),
        .rd_data_1(rDataA1IdEx),
        .rd_data_2(rDataB2IdEx),
        .ALU_op({aluOp1IdEx,aluOp0IdEx}),
        .Opcode_field(opCFieldIdEx),            //input
        .ALU_out(ALU_result_wire),              //output 64 bits
        .Zero(Zero_wire));                      //output 1 bit

//Ex_Mem**********************pipeline register
wire    [202:0] exMemIn_wire, exMemOut_wire;                //203 bits
wire    [63:0]  brAddrExMem, ALUoutExMem, rDataB2ExMem;     //[202:139], [127:64], [63:0]
wire    regWrExMem,memToRegExMem,brExMem,memRdExMem,memWrExMem,ZeroExMem; //[138:133]
wire    [4:0]   imRtExMem;                                  //[132:128]
Ex_Mem_reg EX_MEM_PR_unit (
        .clk(clk),
        .reset(reset),
        .exMemIn({97'b0, exMemIn_wire}),
        .exMemOut(exMemOut_wire));
assign  exMemIn_wire = {brAddrIdEx,                                 //[202:139]
                        regWrIdEx,memToRegIdEx,brIdEx,memRdIdEx,memWrIdEx,Zero_wire, //[138:133]
                        imRtIdEx, ALU_result_wire, rDataB2IdEx }; //[132:128], [127:64], [63:0]

assign  brAddrExMem     = exMemOut_wire[202:139]; //64 bits
assign  regWrExMem      = exMemOut_wire[138];
assign  memToRegExMem   = exMemOut_wire[137];
assign  brExMem         = exMemOut_wire[136];
assign  memRdExMem      = exMemOut_wire[135];
assign  memWrExMem      = exMemOut_wire[134];
assign  ZeroExMem       = exMemOut_wire[133];
assign  imRtExMem       = exMemOut_wire[132:128]; //5 bits
assign  ALUoutExMem     = exMemOut_wire[127:64];
assign  rDataB2ExMem    = exMemOut_wire[63:0];
```

```
    //MEMstg_4================================================================================
    DataMem  DataMem_unit(
            .clk(clk),
            .reset(reset),
            .mem_wr(memWrExMem),
            .mem_rd(memRdExMem),
            .mem_addr(ALUoutExMem[7:0]),
            .wr_data(rDataB2ExMem),         //store, write to mem
            .rd_data(mem_rd_data_wire));    //load, write to register
    //Mem_WB pipeline register
    wire    [134:0] memWbIn_wire, memWbOut_wire; //135 bits
    wire    [63:0]  ALUoutMemWb, memRdDataMemWb;
    //Pipeline_4, Mem/WB
    MEM_WB_reg MEM_WB_PR_unit (
            .clk(clk),
            .reset(reset),
            .memWbIn({165'b0, memWbIn_wire}),
            .memWbOut(memWbOut_wire));
    assign  memWbIn_wire = {regWrExMem, memToRegExMem,imRtExMem, //[134:133], [132:128]
                            ALUoutExMem, mem_rd_data_wire};//[127:64], [63:0]

    assign  regWrMemWb      = memWbOut_wire[134];
    assign  memToRegMemWb   = memWbOut_wire[133];
    assign  imRtMemWb       = memWbOut_wire[132:128];
    assign  ALUoutMemWb     = memWbOut_wire[128:64];
    assign  memRdDataMemWb  = memWbOut_wire[63:0];

    //WBstg_5=================================================================================
    assign wrDataWb = (memToRegMemWb)? memRdDataMemWb : ALUoutMemWb;

endmodule
```