浙江大学

本科实验报告

课程名称: 计算机体系结构

姓　　名: 王俊

学　　院: 海洋学院

专　　业: 海洋工程与技术

学　　号: 3170100186

指导教师: 翁恺

2020 年　11　月　13　日

Lab6—Pipelined CPU resolving control hazards

课程名称：　计算机体系结构　　实验类型：　综合

实验项目名称：　解决控制冲突

学生姓名：　王俊　　专业：　海洋工程与技术　学号：　3170100186

同组学生姓名：　None　　指导老师：　翁恺

实验地点：　曹光彪西-301　　实验日期：　2020　年　1　月　3　日

一、实验目的和要求

Experiment Purpose:

•Understand why and when Control Hazards arise

•Master the methods of resolving Control Hazards

–Freeze or flush

–Predict-not-taken

–Predict-taken

–Delayed-branch

•Master the methods of 1-cycle stall of Predict-not-taken and Predict-taken branch.

•master methods of program verification of Pipelined CPU resolving control hazards

Experiment Apparatus:

- Computer (Intel Core i5 or higher, 4GB RAM or higher) system
- Sword-V4 development board
- Xilinx ISE 14.4 and above development tools

Experimental Materials:

No

二、实验内容和原理

2.1.Experimental task：

•Improve the design of Datapath of 5-stages Pipelined CPU via reducing 3 control stall to 1 stall when using flush

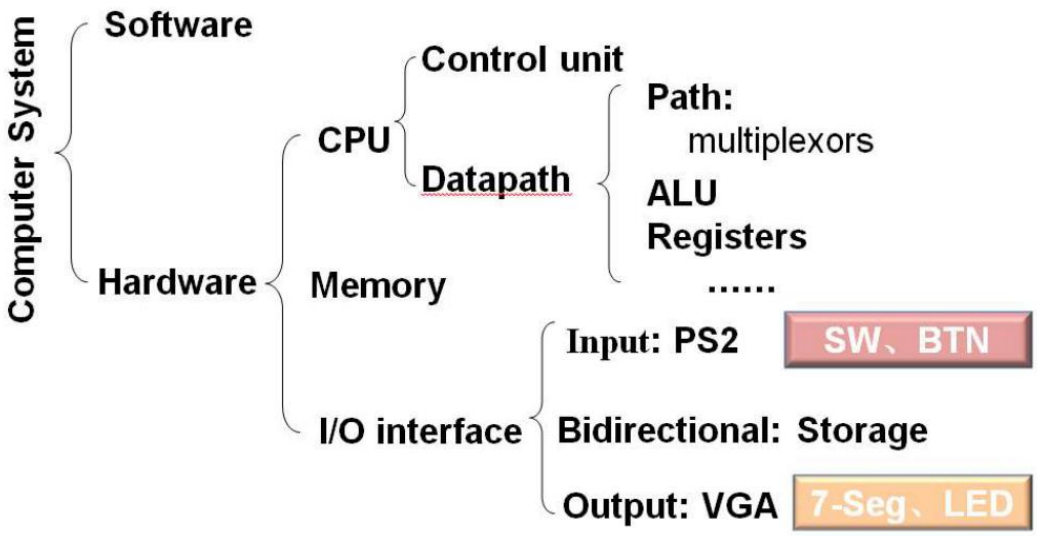
–Bring forward calculation of condition & branch address

–Bring forward bypass unit

•Verify the Pipeline CPU with program and observe the execution of program

2.2.Basic principle

2.2.1.Computer system decomposition



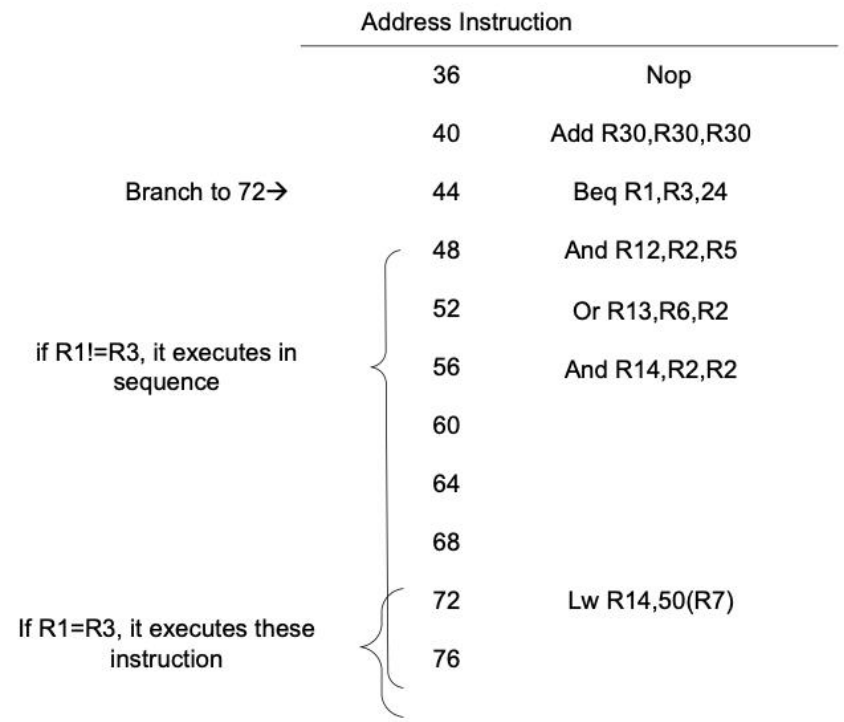
2.2.2.Definition of control hazard:

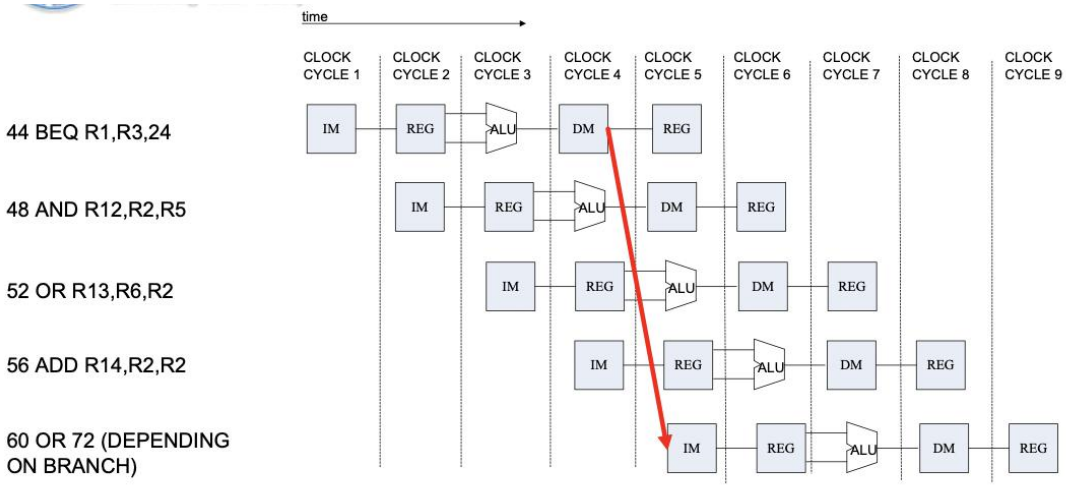
•Control Hazards arise from the pipelining of branches and other instructions that change the PC.

•Control hazards can cause a greater performance loss for our MIPS pipeline compared with data hazards.

•Reducing Pipeline Branch Penalties.

2.2.3.Instruction Demo & Execution Result





2.2.4.Methods of resolving Control hazards

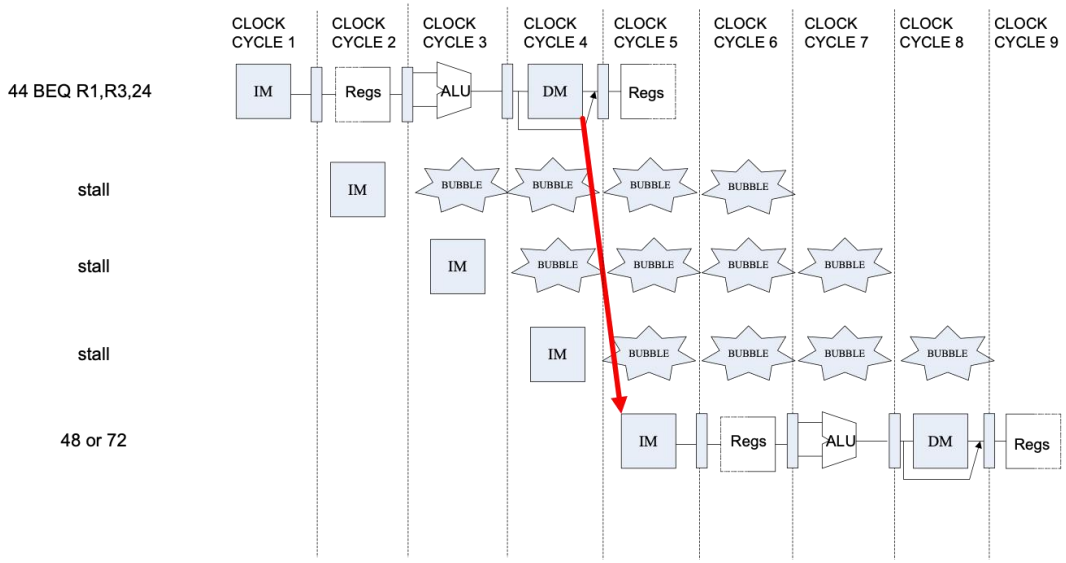
•Freeze or flush the pipeline

•Predict-not-taken

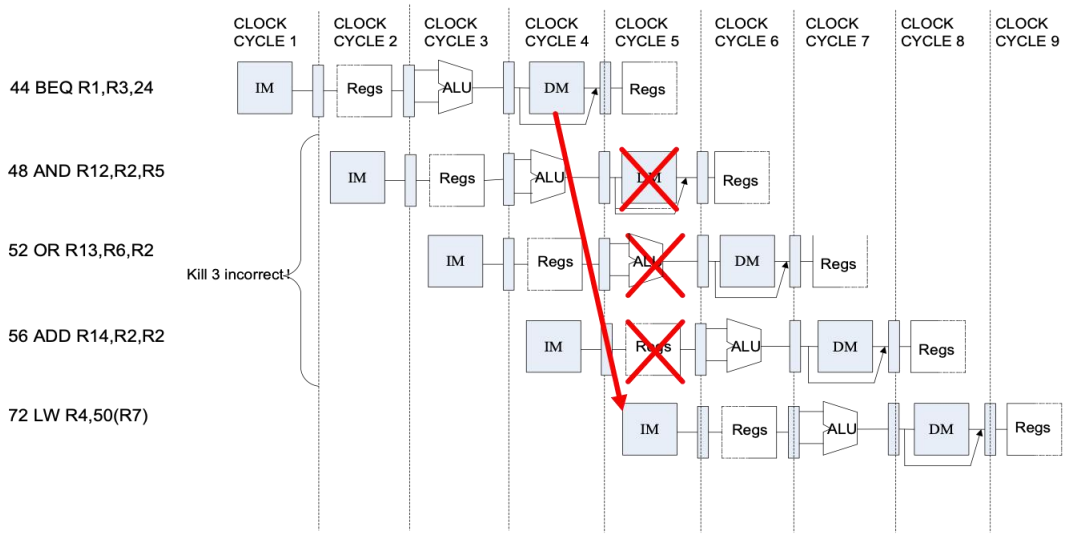
•Predict-taken

•Delayed branch

Freeze method:



Predict-not-taken(not in lab) :

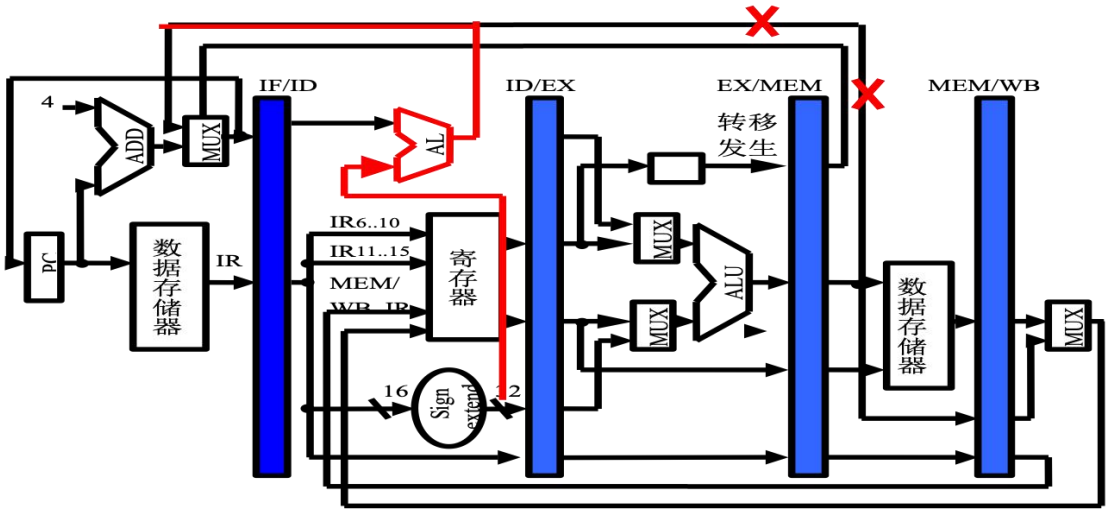


How to implement it ?

•Normally: pc←pc+4, no stall whenever ((exop or memop)== branch)

•If mem_op=branch and condition_code==1, then Kill the following 3 instructions flying in pipeline.

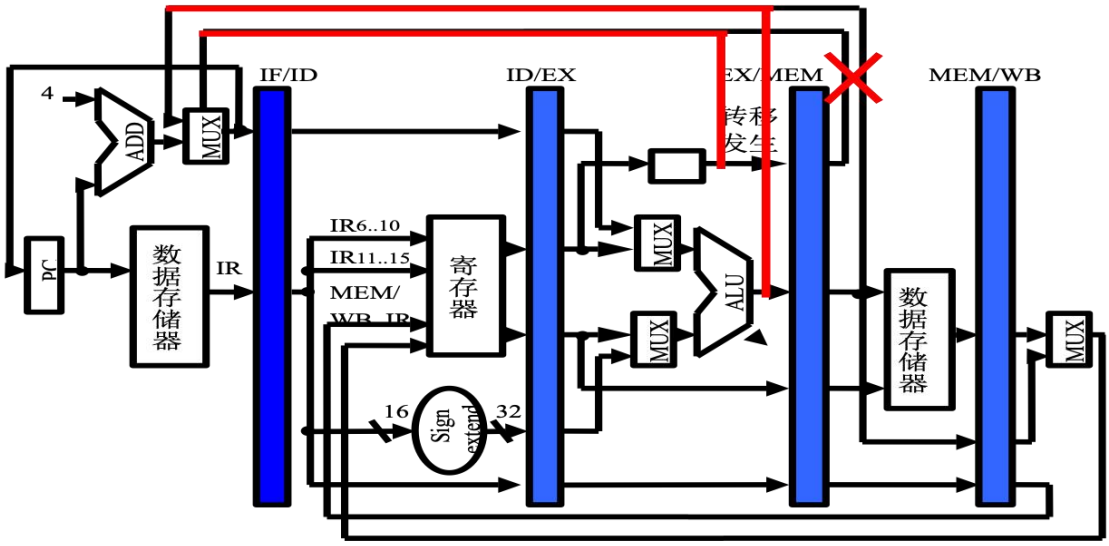
Predict taken only when branch target can be got earlier that condition code : However there's no advantage for MIPS pipeline, but if the datapath can be changed as following.



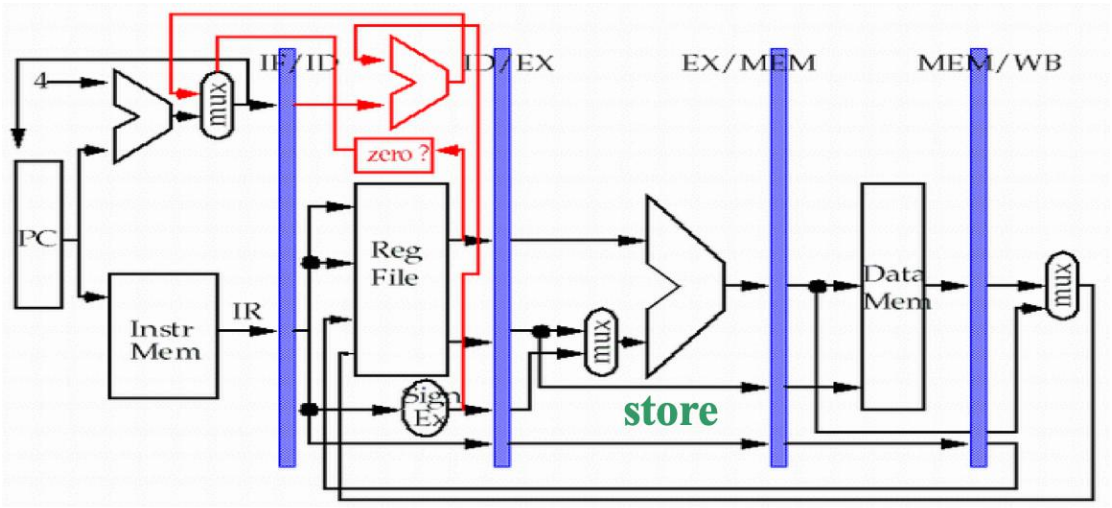
三、实验过程

3.1.Reduce the control stall by calculate target and cc asap.

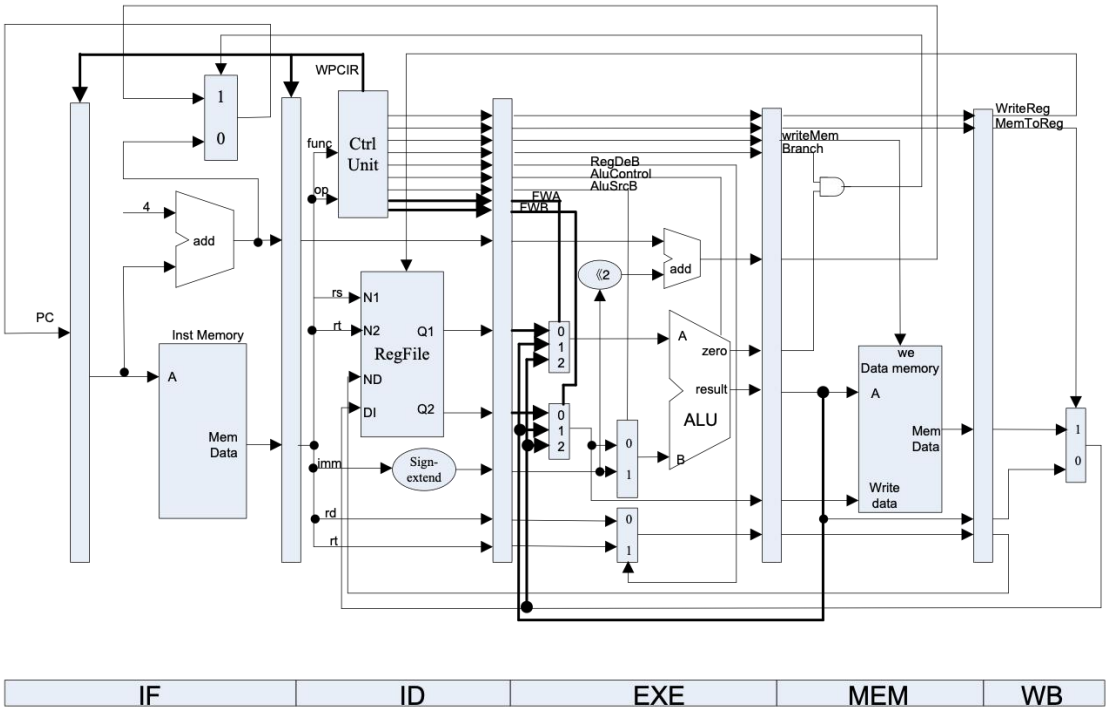
•Bring forward calculation of branch condition from MEM Stage to EX Stage, stall reduce from 3-cycle to 2-cycle.



•Then bring forward from EX to ID, stall reduce from 2-cycle to 1-cycle.



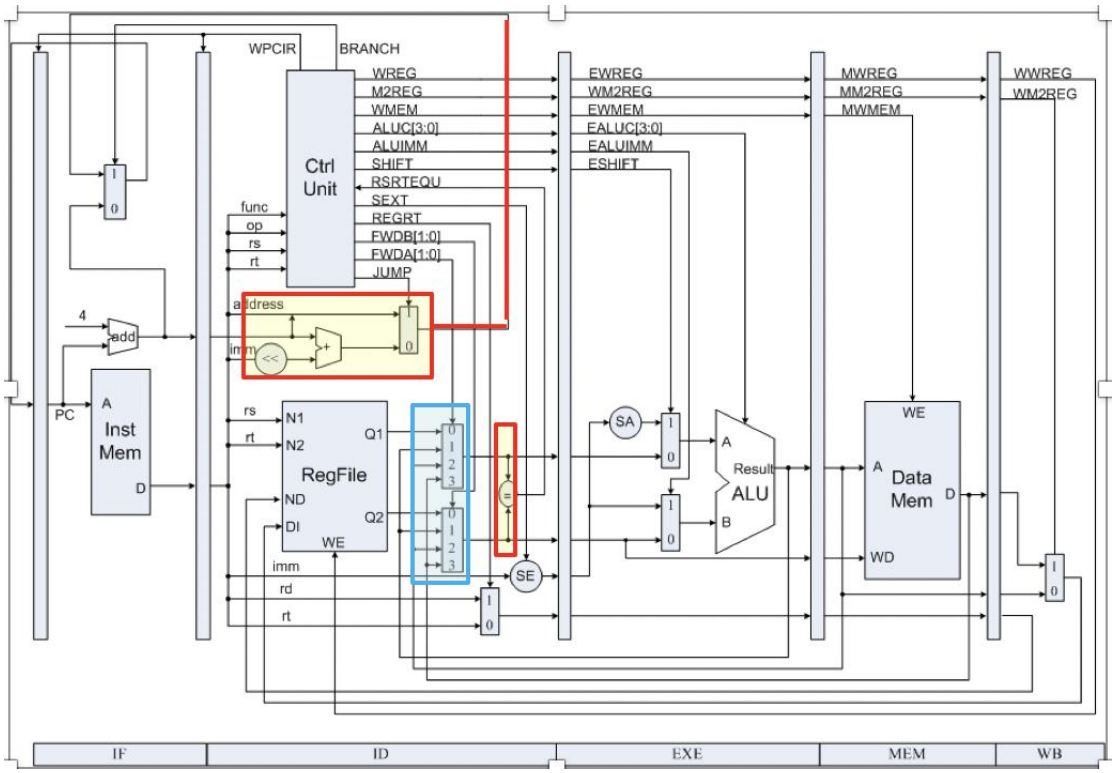
What we have done by far:



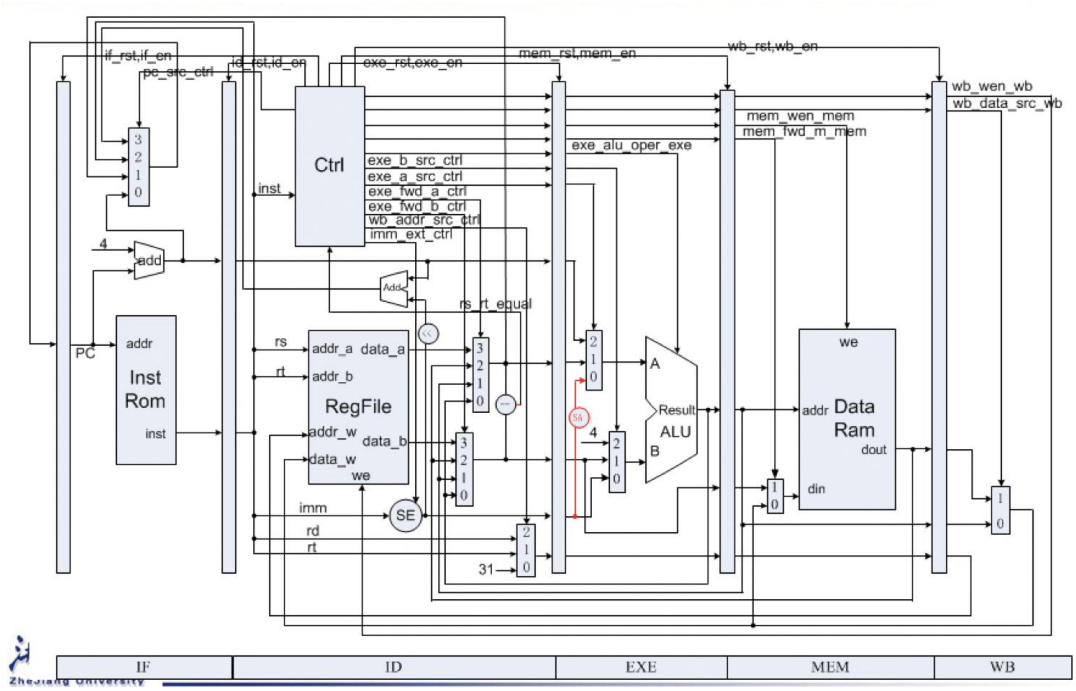
The control logic:

- Stall
  - Control stall: ex_op / mem_op is branch or jmp. (Flush)
  - Load stall: (ex_op=lw) & ((ex_rd=id_rs) & exist_id_rs)
    or ((ex_rd= id_rt) & exist_id_rt)
- FWA:
  - 1: (ex_op=ALUop) && ((ex_rd=id_rs) & exist_id_rs)
  - 2: (mem_op=ALUop/LWop) && ((mem_rd=id_rs) & exist_id_rs)
  - 0:
- FWB:
  - 1: (ex_op=ALUop) && ((ex_rd=id_rt) & exist_id_rt)
  - 2: (mem_op=ALUop/LWop) && ((mem_rd=id_rt) & exist_id_rt)
  - 0:

Datapath resolving Control Hazards :

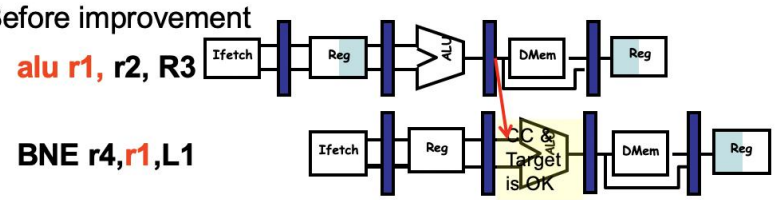


My design:

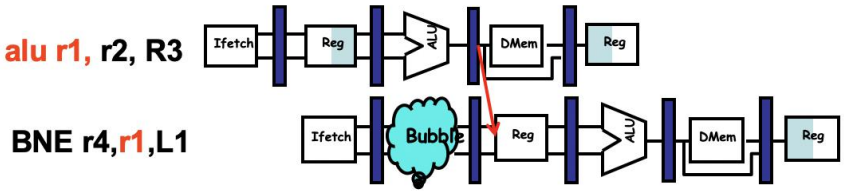


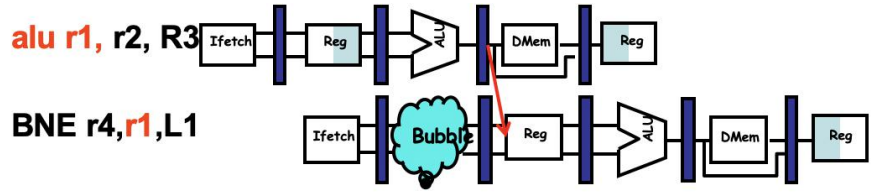
3.2.Move the forwarding control logic to ID stage

Before improvement

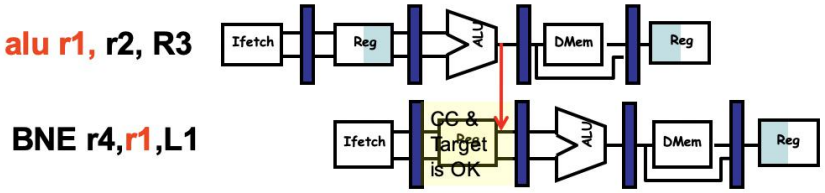


What if CC and target is calculated in ID stage ?

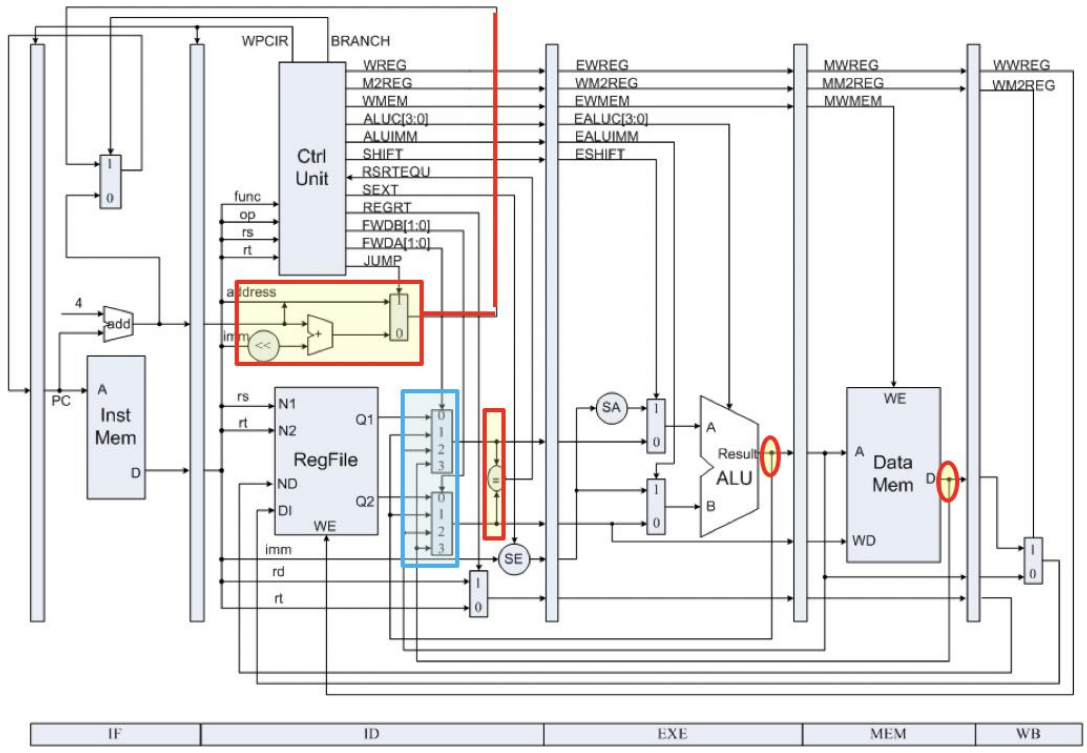




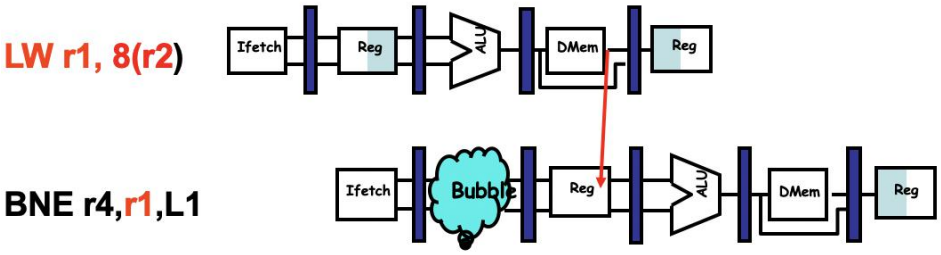
Move the forwarding control logic into ID stage,
and Also, send out ALUoutput and LWDR earlier before the latch.



Move forward the forwarding control logic & send out ALUoutput/LWDR before the latch.:



But new stall still arise:



- So what about your stall logic and forwarding control logic ?

Forward 提前:

```
always @(*) begin
    if (id_en) begin
        case (ForwardA)
            FORWARD_NORMAL:    data_rs_fwd = data_rs;
            FORWARD_ADD_ADD:   data_rs_fwd = alu_out_exe;
            FORWARD_ADD_X_ADD:data_rs_fwd = regw_data_mem;
            FORWARD_LW_ADD:    data_rs_fwd = alu_out_mem;
        endcase
        case (ForwardB)
            FORWARD_NORMAL:    data_rt_fwd = data_rt;
            FORWARD_ADD_ADD:   data_rt_fwd = alu_out_exe;
            FORWARD_ADD_X_ADD:data_rt_fwd = regw_data_mem;
            FORWARD_LW_ADD:    data_rt_fwd = alu_out_mem;
        endcase
    end
end
```

```
assign mem_dout = (fwd_m_mem) ? regw_data_wb : data_rt_mem;
```

In the controller:

```
// forwarding
    reg load_stall;

    always @(*) begin
        exe_fwd_a = 3;
        exe_fwd_b = 3;
        load_stall = 0;
        fwd_m = 0;

        if(wb_wen_exe && regw_addr_exe != 0) begin
            if(rs_used && regw_addr_exe == addr_rs) begin
                // read rs next to LW, have to stall 1 cycle.
                if(is_load_exe)
                    load_stall = 1;
                // read ALU result
                else
                    exe_fwd_a = 0;
            end
            else if(rt_used && regw_addr_exe == addr_rt) begin
                // read rt next to LW, 2 possibilities
                if(is_load_exe) begin
                    // SW next to LW, no EX phase
                    if(is_store)
                        fwd_m = 1;
                    // other instructions next to LW, have EX phase
                    else
                        load_stall = 1;
                end
                // read ALU result
                else
                    exe_fwd_b = 0;
            end
        end
```

```
    if(wb_wen_mem && regw_addr_mem != 0) begin
        if(rs_used && regw_addr_mem == addr_rs) begin
            // data read from memory(after 1 cycle stall)
            if(is_load_mem)
                exe_fwd_a = 2;
            // ALU result
            else
                exe_fwd_a = 1;
        end
        else if(rt_used && regw_addr_mem == addr_rt) begin
            if(is_load_mem && ~is_store)
                exe_fwd_b = 2;
            else
                exe_fwd_b = 1;
        end
    end

    // pass
end
```

new branch computation:

```
always @(*) begin
    is_branch_id <= (pc_src_ctrl != PC_NEXT && branch_target_id != inst_addr_next_id);
end

always @(*) begin
    case (pc_src_ctrl)
        PC_JUMP: branch_target_id <= {inst_addr_next_id[31:28], inst_data_id[25:0], 2'b0};
        PC_JR: branch_target_id <= data_rs_id;
        PC_BEQ: branch_target_id <= (data_rs_id==data_rt_id)? (inst_addr_next_id + {data_imm[29:0], 2'b0}) :inst_addr_next_id;
        PC_BNE: branch_target_id <= (data_rs_id!=data_rt_id)? (inst_addr_next_id + {data_imm[29:0], 2'b0}): inst_addr_next_id;
        default: branch_target_id <= inst_addr_next_id;  // will never used
    endcase
end
```

3.3. Predict-not-taken

If rs_rt_equal 提前到 ID 阶段:

```
assign
    rs_rt_equal_id = (data_rs_fwd == data_rt_fwd);
```

Branch_ctrl 提前到 ID:

```
always @(*) begin
    regw_addr_id = inst_data_id[15:11];
    case (wb_addr_src_ctrl)
        WB_ADDR_RD: regw_addr_id = addr_rd; // write back address
        WB_ADDR_RT: regw_addr_id = addr_rt;
        WB_ADDR_LINK: regw_addr_id = GPR_RA;  // write back to $ra
    endcase
end
```

ID 中的 forward：

```
always @(*) begin
   if (id_en) begin
      case (ForwardA)
         FORWARD_NORMAL:    data_rs_fwd = data_rs;
         FORWARD_ADD_ADD:   data_rs_fwd = alu_out_exe;
         FORWARD_ADD_X_ADD:data_rs_fwd = regw_data_mem;
         FORWARD_LW_ADD:    data_rs_fwd = alu_out_mem;
      endcase
      case (ForwardB)
         FORWARD_NORMAL:    data_rt_fwd = data_rt;
         FORWARD_ADD_ADD:   data_rt_fwd = alu_out_exe;
         FORWARD_ADD_X_ADD:data_rt_fwd = regw_data_mem;
         FORWARD_LW_ADD:    data_rt_fwd = alu_out_mem;
      endcase
   end
end
```

Controller 中获得 pc_src:

```
INST_BEQ: begin
   pc_src = rs_rt_equal ? PC_BEQ : PC_NEXT;
   exe_a_src = EXE_A_BRANCH;
   exe_b_src = EXE_B_BRANCH;
   exe_alu_oper = EXE_ALU_ADD;
   imm_ext = 1;
   rs_used = 1;
   rt_used = 1;
end
INST_BNE: begin
   pc_src = rs_rt_equal ? PC_NEXT : PC_BNE;  //change
   exe_a_src = EXE_A_BRANCH;
   exe_b_src = EXE_B_BRANCH;
   exe_alu_oper = EXE_ALU_ADD;
   imm_ext = 1;
   rs_used = 1;
   rt_used = 1;
```

从ID获取

根据rs_rt_equal 输出 PC_src

IF taken: branch_stall = 1

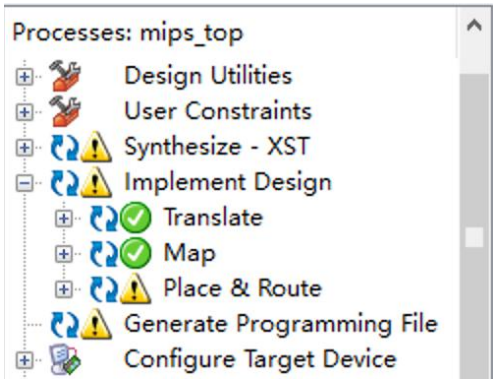
```
always @(*) begin
   branch_stall = 0;
   if (pc_src != PC_NEXT)
      branch_stall = 1;
end
```

Then, refresh the id:

```
else if (branch_stall) begin
    id_rst = 1;
end
```

四、实验结果分析

编译 bit 文件：



Use the code given:

```
【注释版】
begin:
0       Lw $1, 20($zero)                //R1=4
1       Lw $2, 24($zero)                //R2=1
2       Add $3,$2,$1            //R3=5  //2LW-ALU:forwarding:1 stall
3       Sub $4,$3,$1            //R4=1  //2ALU-ALU
4       And $5,$3,$1            //R5=4  //无冲突
5       Or $6,$3,$1             //R6=5  //无冲突
6       addi $6,$3,4            //$6=9  //无冲突
```

Put the hex file in the current folder and have a test：

data_mem.hex

```
00000000
00000008
00000000
00000000
00000000
00000004
00000001
```

inst_mem.hex

```
8C010014
8C020018
00411820
00612022
00612824
00613025
20660004
00013820
8CE80000
ACE80008
8CE90008
AD270000
8D2A0000
00215820
002A5020
00225020
114B0002
8CE10008
8C020018
00411820
00612022
20940001
34948000
14220001
8C010014
8C020018
00411820
00612022
0C000021
00012882
```

Vga:

When it comes to, not taken: no nop; if taken: 1 nop

```
f       Add $10,$1,$2           //R10=5
10      Beq $10, $11 ,lable1    // not taken//2+1ALU-BEQ//      branch x → 不停
11      Lw $1, 8($7)            //R1=8
12      Lw $2, 24($zero)                //R2=1
lable1:
13      Add $3,$2,$1            //R3=9  //2LW-ALU
14      Sub $4,$3,$1            //R4=1   //2ALU-R-R
15      Addi $20, $4, 1         //R20=2 //2ALU-addi
16      Ori $20, $4, 0x8000     //R20=0x8001    //无冲突 无符号扩展
17      Bne $1, $2, lable2              //taken x          → 1 nop
18      Lw $1,20($zero)         //不执行
lable2:
19      Lw $2,24($zero)         //R2=1
1a      Add $3,$2,$1            //R3=9   //2ALU
```

And the result is the same as what we predict, so the control hazard is solving!