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

Lab4—Pipelined CPU with stall

课程名称: 计算机体系结构 实验类型: 综合

实验项目名称: 用 stall 解决数据冲突

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 11 月 13 日

一、实验目的和要求

Experiment Purpose:

•Understand the principles of Pipelined CPU Stall

•Understand the principles of Data hazard

•Master the method of Pipelined CPU Stalls Detection and Stall the Pipeline.

•master methods of program verification of Pipelined CPU with Stall

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:

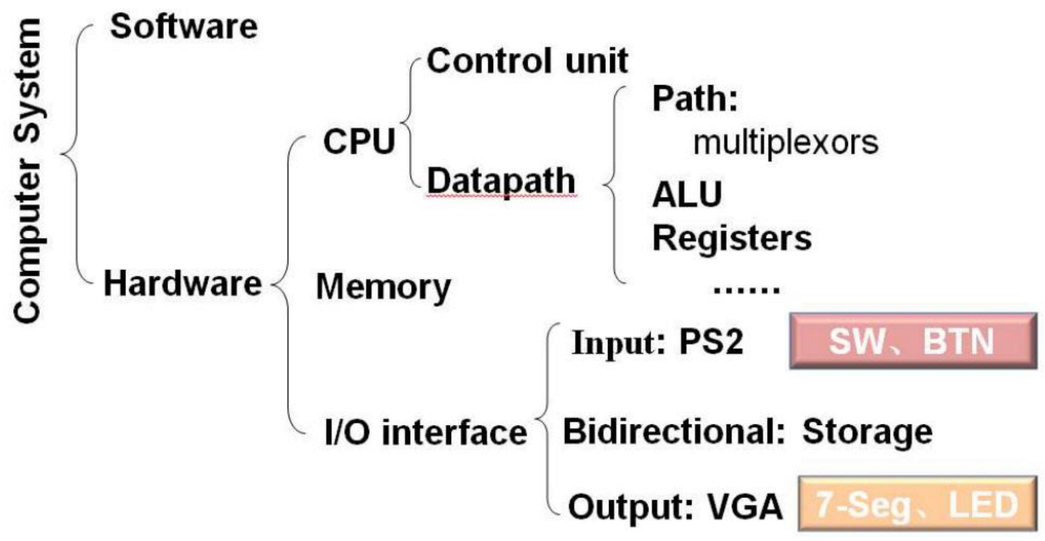
•Design the Stall Part of Datapath of 5-stages Pipelined CPU

•Modify the CPU Controller, adding Condition Detection of Stall.

•Verify the Pp. CPU with program and observe the execution of program

2.2.Basic principle

2.2.1.Computer system decomposition



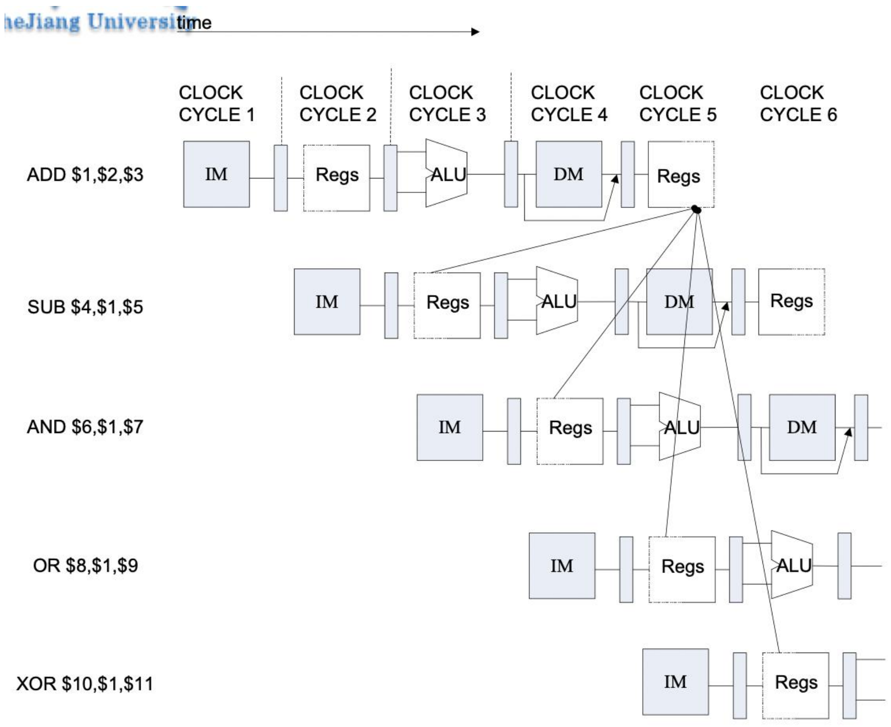
2.2.2.Definition of data hazard:

•Data Hazards arise when an instruction depends on the results of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

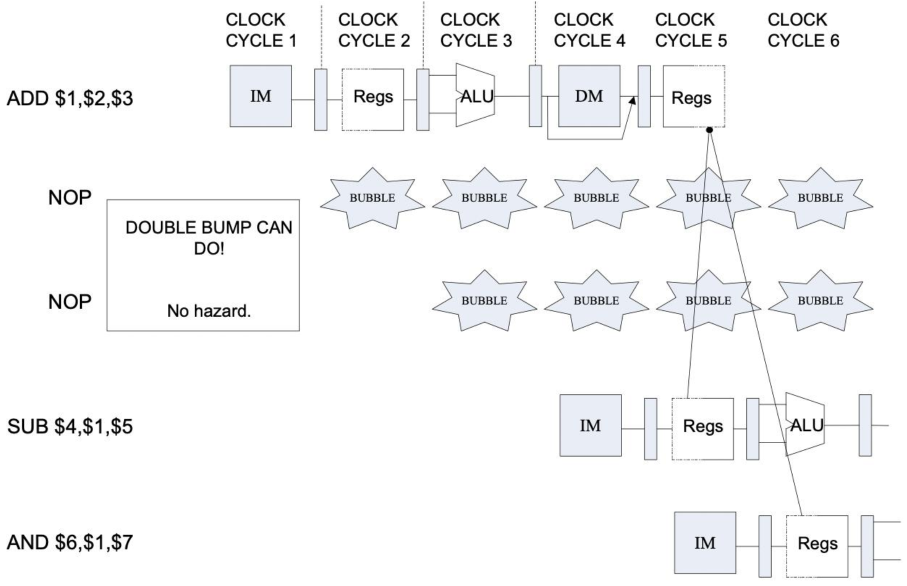
•When inst. i executes before instr. j, there are data hazards:

- •RAW, i.e. instr. j read a source before instr. i writes it.
- •WAW, i.e. instr. j write an operand before instr. i writes it.
- •WAR, i.e. instr. j write a destination before instr. i read it.

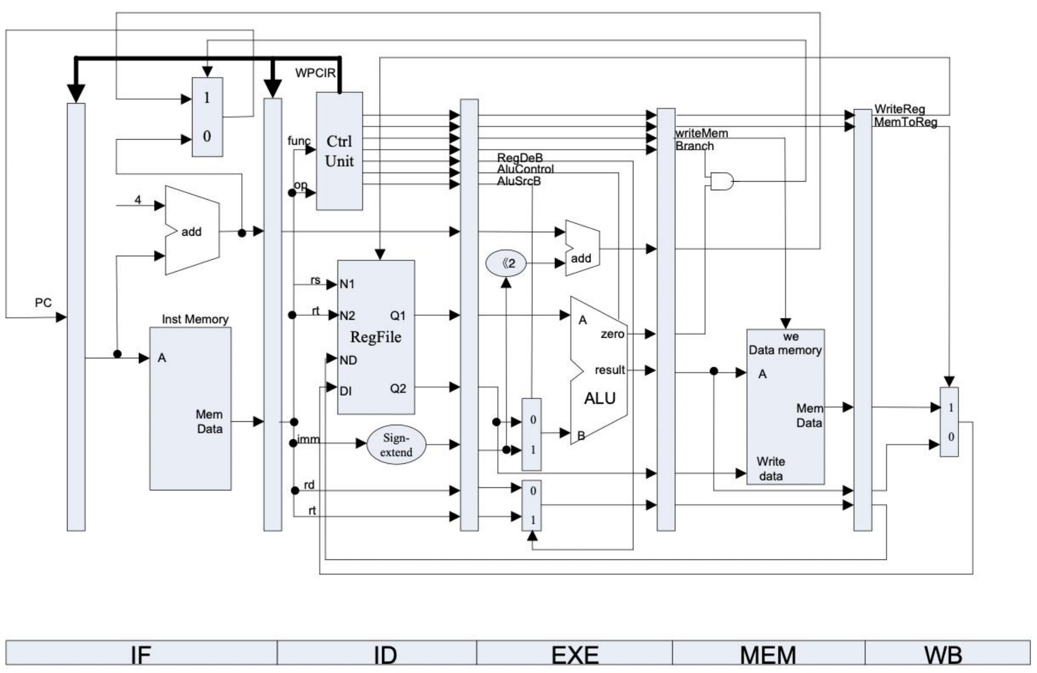
Instruction demo:



Data Hazard Causes Stalls:

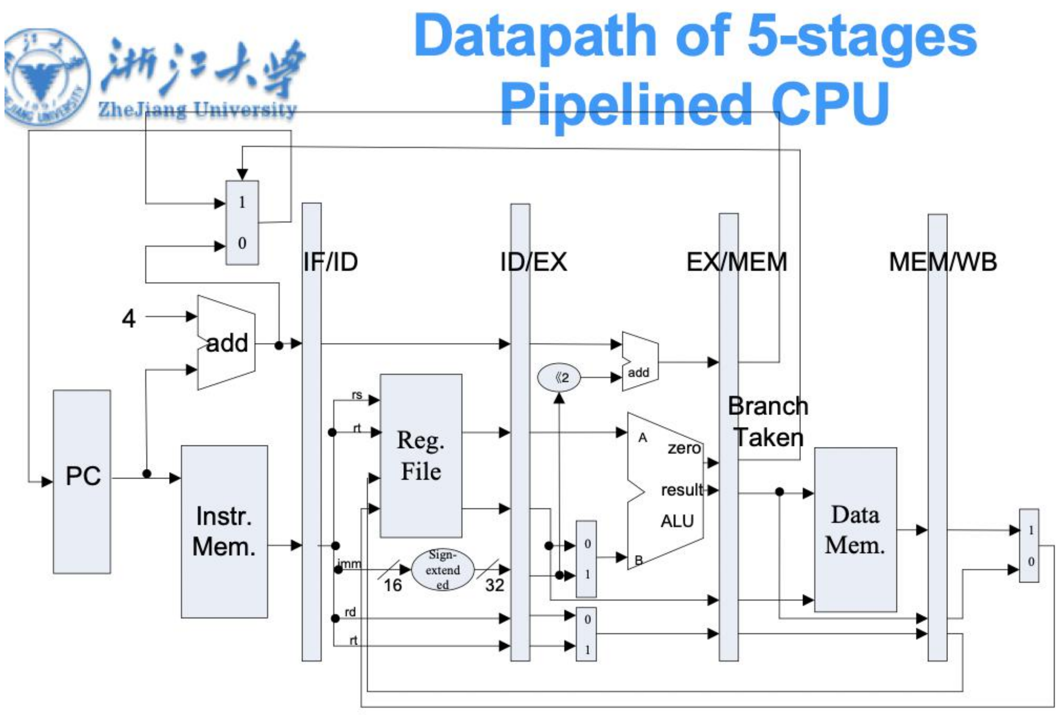


And how to Stall the Pipeline?

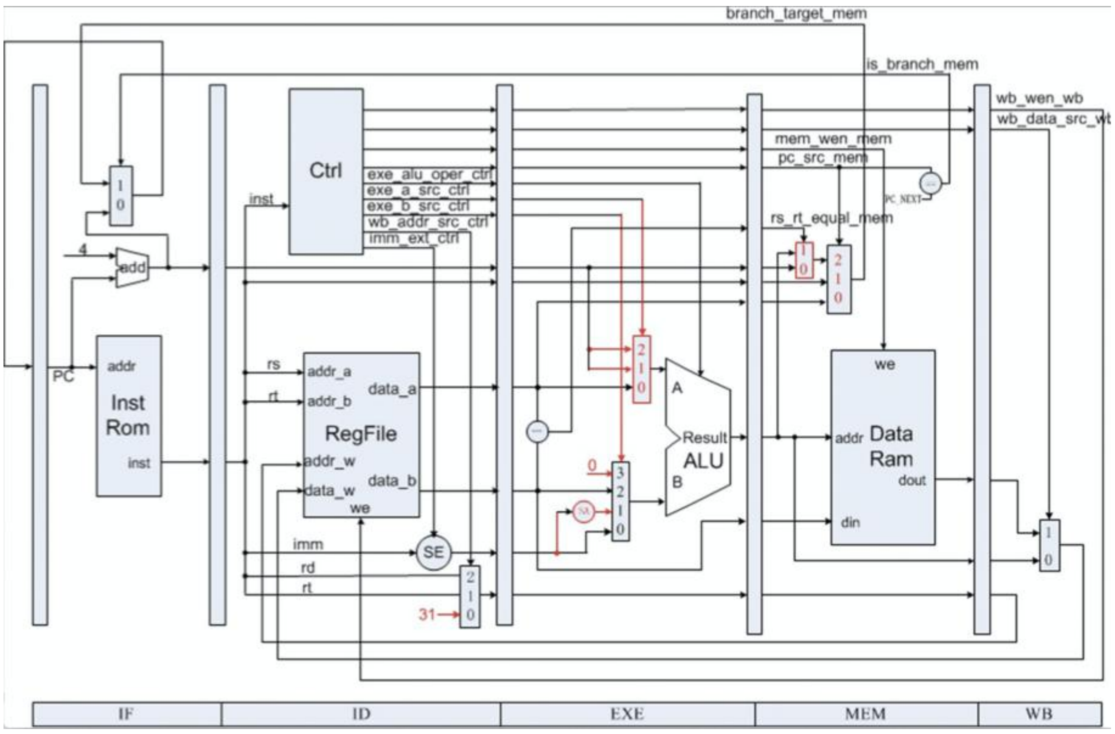


2.2.3.Datapath of 5-stages pipelined CPU

- **The datapath from the slides:**

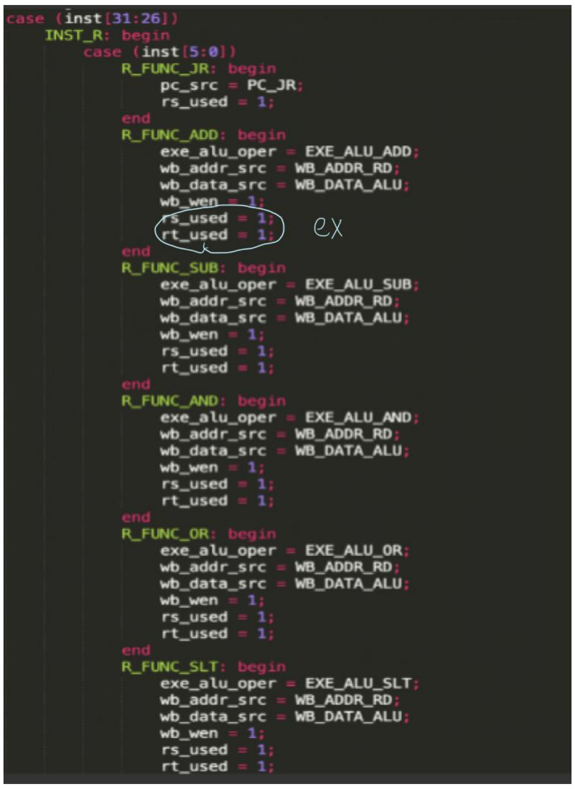


- **My design:**



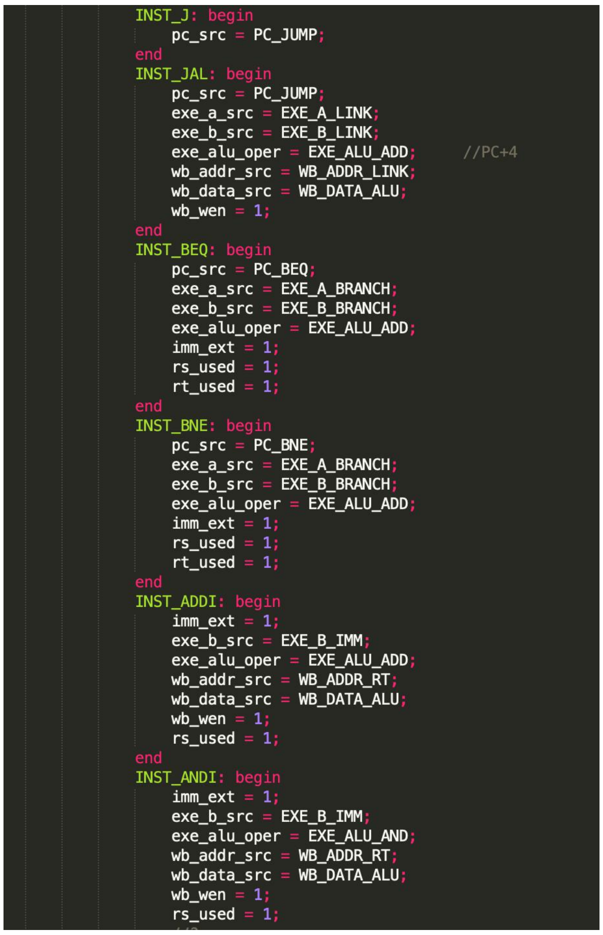
三、实验过程

3.1.Add rt-used and rs-used in the instructions:

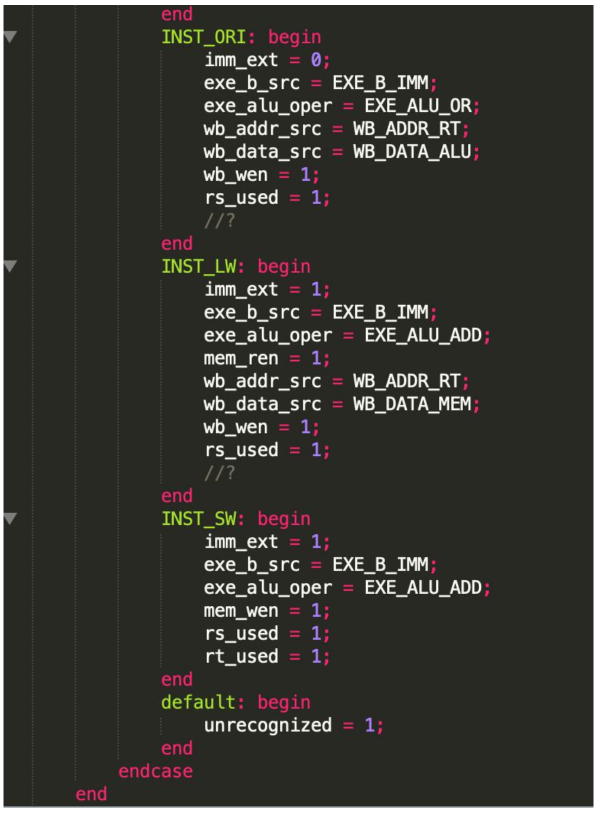


```
case (inst[31:26])
    INST_R: begin
        case (inst[5:0])
            R_FUNC_JR: begin
                pc_src = PC_JR;
                rs_used = 1;
            end
            R_FUNC_ADD: begin
                exe_alu_oper = EXE_ALU_ADD;
                wb_addr_src = WB_ADDR_RD;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                rt_used = 1;
            end
            R_FUNC_SUB: begin
                exe_alu_oper = EXE_ALU_SUB;
                wb_addr_src = WB_ADDR_RD;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                rt_used = 1;
            end
            R_FUNC_AND: begin
                exe_alu_oper = EXE_ALU_AND;
                wb_addr_src = WB_ADDR_RD;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                rt_used = 1;
            end
            R_FUNC_OR: begin
                exe_alu_oper = EXE_ALU_OR;
                wb_addr_src = WB_ADDR_RD;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                rt_used = 1;
            end
            R_FUNC_SLT: begin
                exe_alu_oper = EXE_ALU_SLT;
                wb_addr_src = WB_ADDR_RD;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                rt_used = 1;
```

ex



```
            INST_J: begin
                pc_src = PC_JUMP;
            end
            INST_JAL: begin
                pc_src = PC_JUMP;
                exe_a_src = EXE_A_LINK;
                exe_b_src = EXE_B_LINK;
                exe_alu_oper = EXE_ALU_ADD;     //PC+4
                wb_addr_src = WB_ADDR_LINK;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
            end
            INST_BEQ: begin
                pc_src = PC_BEQ;
                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 = PC_BNE;
                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_ADDI: begin
                imm_ext = 1;
                exe_b_src = EXE_B_IMM;
                exe_alu_oper = EXE_ALU_ADD;
                wb_addr_src = WB_ADDR_RT;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
            end
            INST_ANDI: begin
                imm_ext = 1;
                exe_b_src = EXE_B_IMM;
                exe_alu_oper = EXE_ALU_AND;
                wb_addr_src = WB_ADDR_RT;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
```



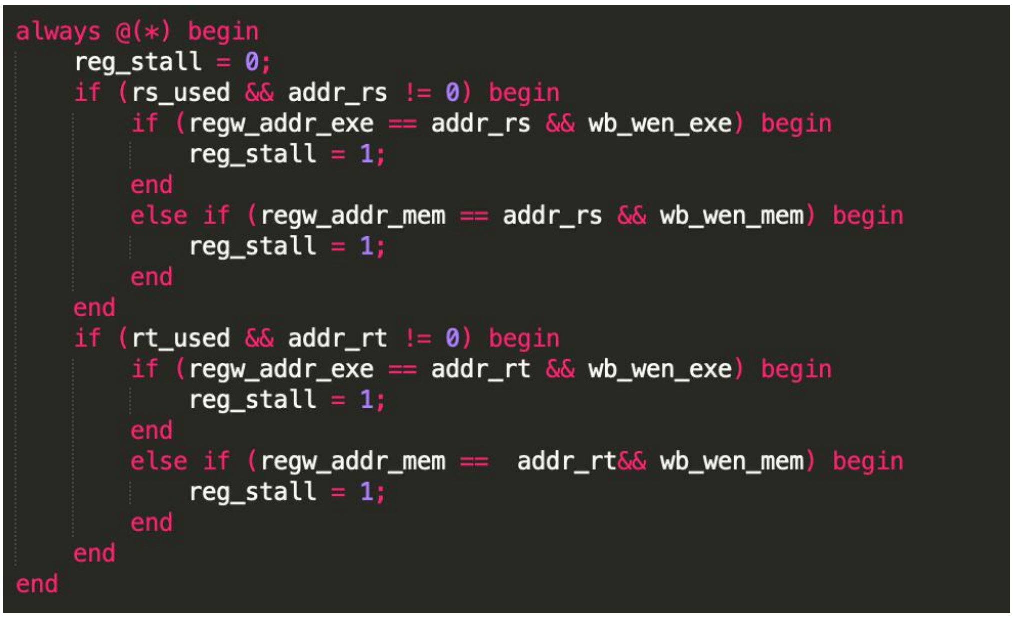
```
            end
            INST_ORI: begin
                imm_ext = 0;
                exe_b_src = EXE_B_IMM;
                exe_alu_oper = EXE_ALU_OR;
                wb_addr_src = WB_ADDR_RT;
                wb_data_src = WB_DATA_ALU;
                wb_wen = 1;
                rs_used = 1;
                //?
            end
            INST_LW: begin
                imm_ext = 1;
                exe_b_src = EXE_B_IMM;
                exe_alu_oper = EXE_ALU_ADD;
                mem_ren = 1;
                wb_addr_src = WB_ADDR_RT;
                wb_data_src = WB_DATA_MEM;
                wb_wen = 1;
                rs_used = 1;
                //?
            end
            INST_SW: begin
                imm_ext = 1;
                exe_b_src = EXE_B_IMM;
                exe_alu_oper = EXE_ALU_ADD;
                mem_wen = 1;
                rs_used = 1;
                rt_used = 1;
            end
            default: begin
                unrecognized = 1;
            end
        endcase
    end
```

3.2.To detect data hazard and implement the reg_stall:

- **If detect the data hazard, use reg_stall:**

Ex For rs:

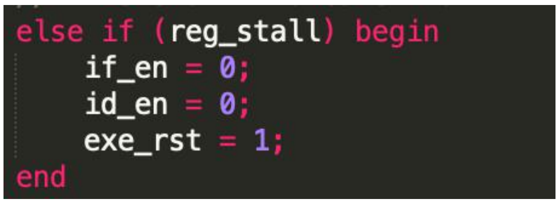
if rs is needed to be used && the regw_addr is still in the exe and mem stage && there is a write back signal: reg_stall = 1



```
always @(*) begin
    reg_stall = 0;
    if (rs_used && addr_rs != 0) begin
        if (regw_addr_exe == addr_rs && wb_wen_exe) begin
            reg_stall = 1;
        end
        else if (regw_addr_mem == addr_rs && wb_wen_mem) begin
            reg_stall = 1;
        end
    end
    if (rt_used && addr_rt != 0) begin
        if (regw_addr_exe == addr_rt && wb_wen_exe) begin
            reg_stall = 1;
        end
        else if (regw_addr_mem ==  addr_rt&& wb_wen_mem) begin
            reg_stall = 1;
        end
    end
end
```

- **implement the reg_stall:**

this stall indicate that ID is waiting for previous instruction, should insert NOPs between ID and EXE:

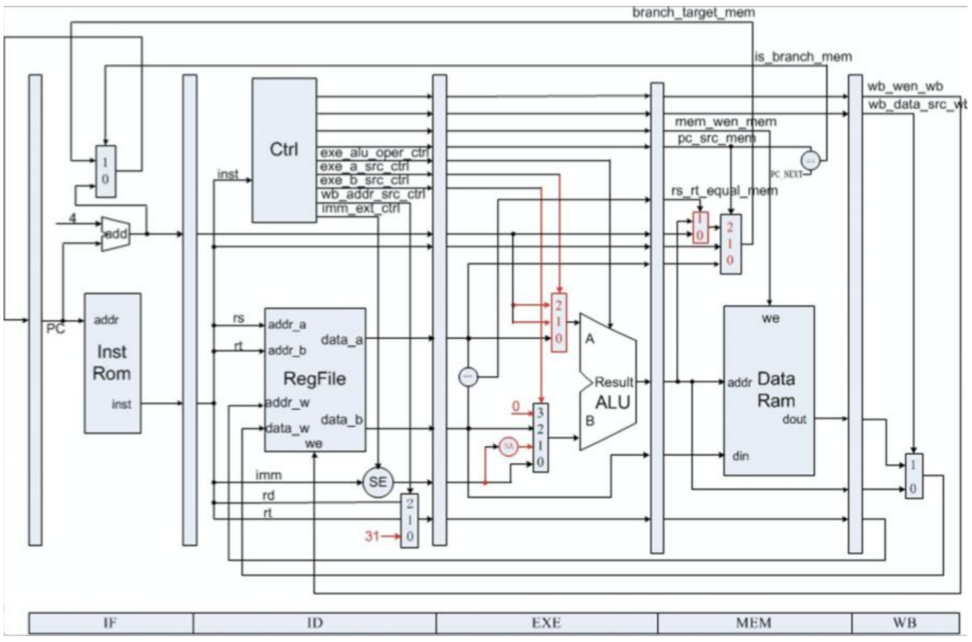


```
else if (reg_stall) begin
    if_en = 0;
    id_en = 0;
    exe_rst = 1;
end
```

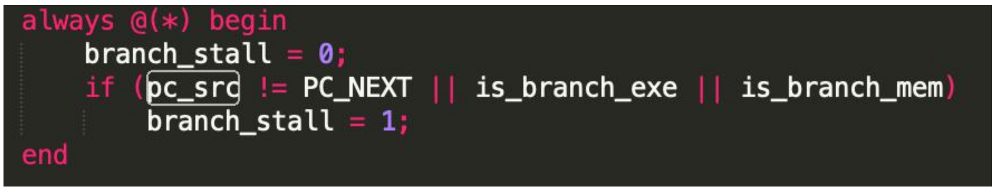
3.3.To detect control hazard:

The judge is in the MEM stage:

whether the pc_next is equal to pc_src to decide whether is taken or not?



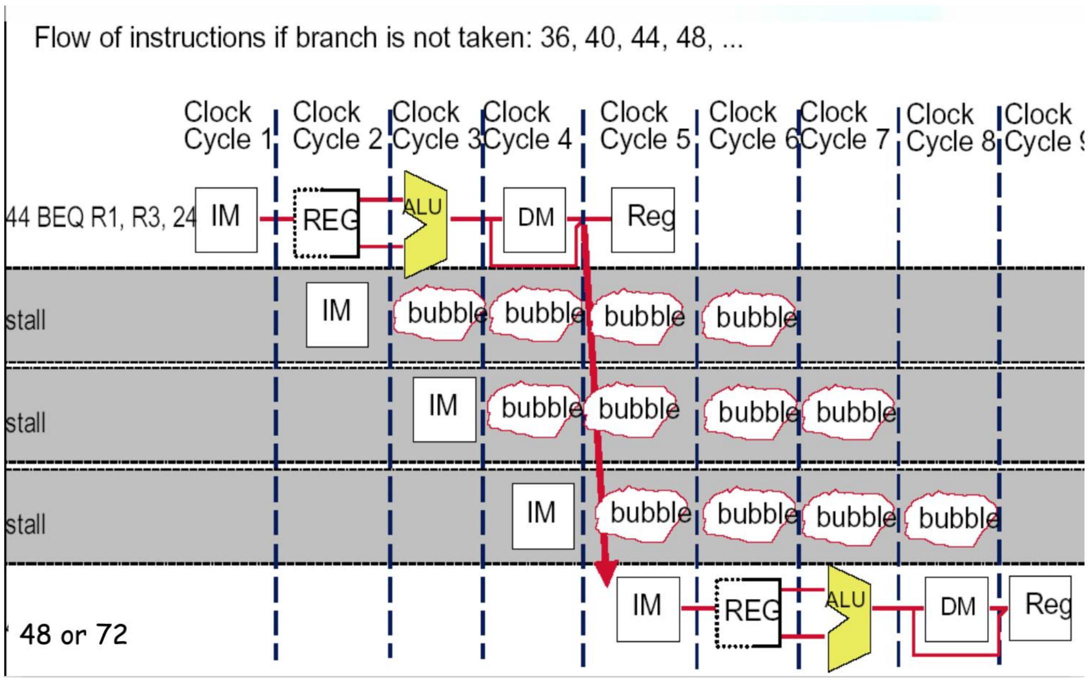
Condition:



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

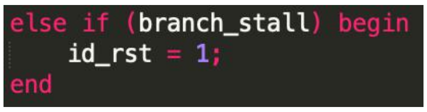
// this stall indicate that a jump/branch instruction is running, so that

3 NOP should be inserted between IF and ID:



Flow of instructions if branch is not taken: 36, 40, 44, 48, ...

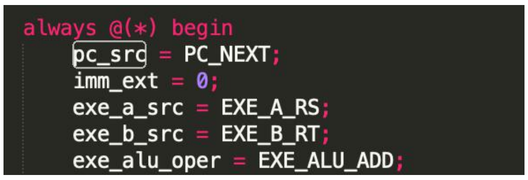
Branch-stall:



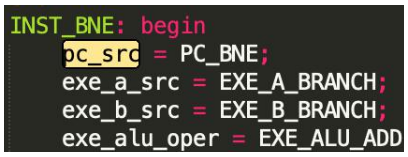
```
else if (branch_stall) begin
    id_rst = 1;
end
```

Use pc_src to judge whether is a branch instruction:

If it is not a branch instruction, pc_src = pc_NEXT

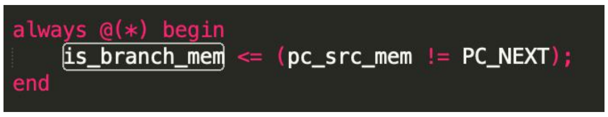


```
always @(*) begin
    pc_src = PC_NEXT;
    imm_ext = 0;
    exe_a_src = EXE_A_RS;
    exe_b_src = EXE_B_RT;
    exe_alu_oper = EXE_ALU_ADD;
```

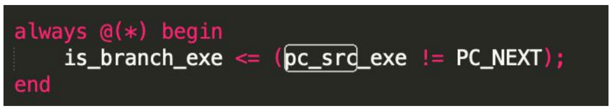


```
INST_BNE: begin
    pc_src = PC_BNE;
    exe_a_src = EXE_A_BRANCH;
    exe_b_src = EXE_B_BRANCH;
    exe_alu_oper = EXE_ALU_ADD
```

To judge whether is a branch instruction in exe and mem stage;



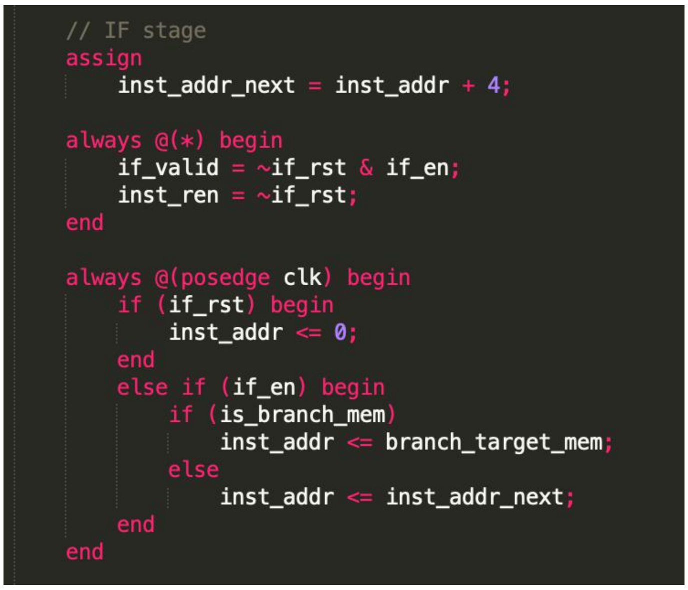
```
always @(*) begin
    is_branch_mem <= (pc_src_mem != PC_NEXT);
end
```



```
always @(*) begin
    is_branch_exe <= (pc_src_exe != PC_NEXT);
end
```

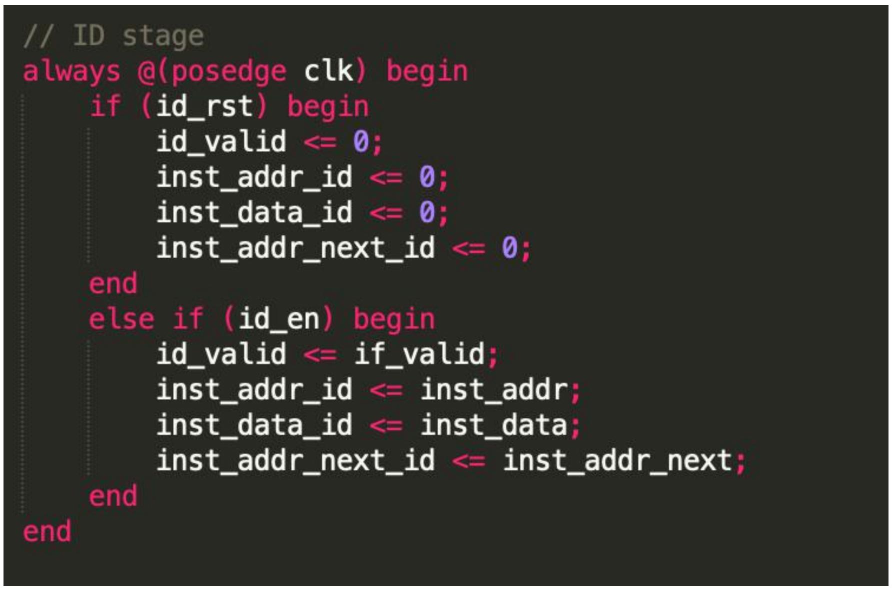
3.4.Each stages:

IF stage with en:



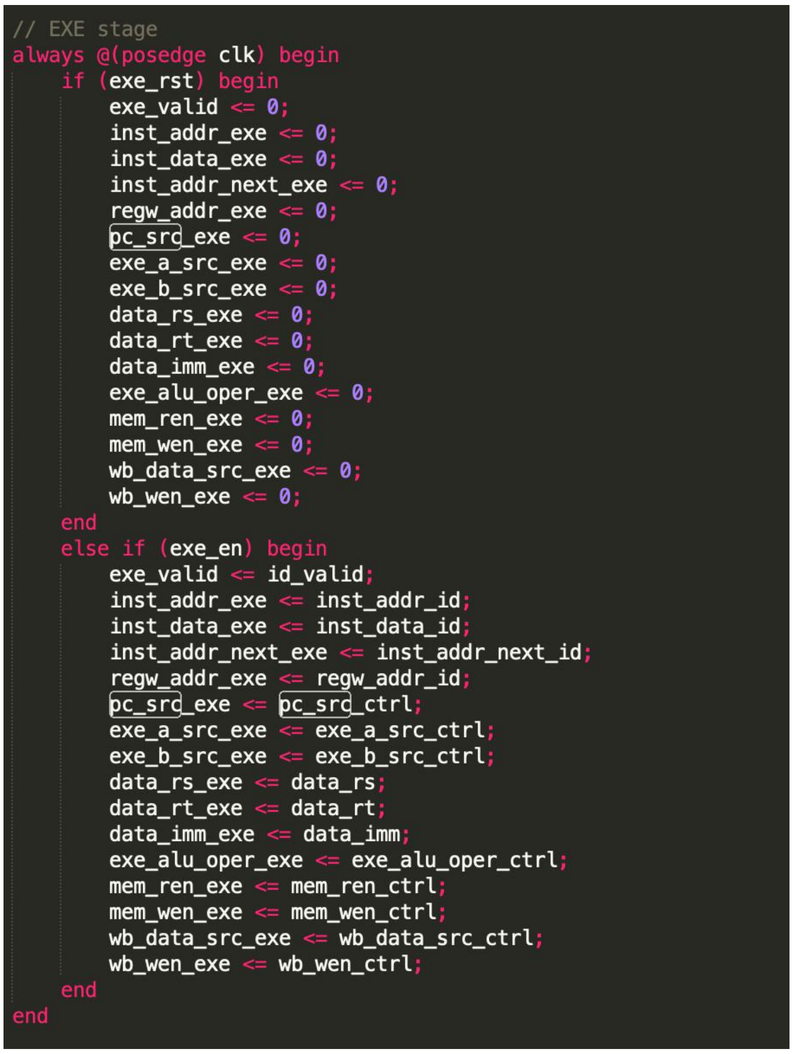
```
// IF stage
assign
    inst_addr_next = inst_addr + 4;

always @(*) begin
    if_valid = ~if_rst & if_en;
    inst_ren = ~if_rst;
end

always @(posedge clk) begin
    if (if_rst) begin
        inst_addr <= 0;
    end
    else if (if_en) begin
        if (is_branch_mem)
            inst_addr <= branch_target_mem;
        else
            inst_addr <= inst_addr_next;
    end
end
```

ID stage with en:



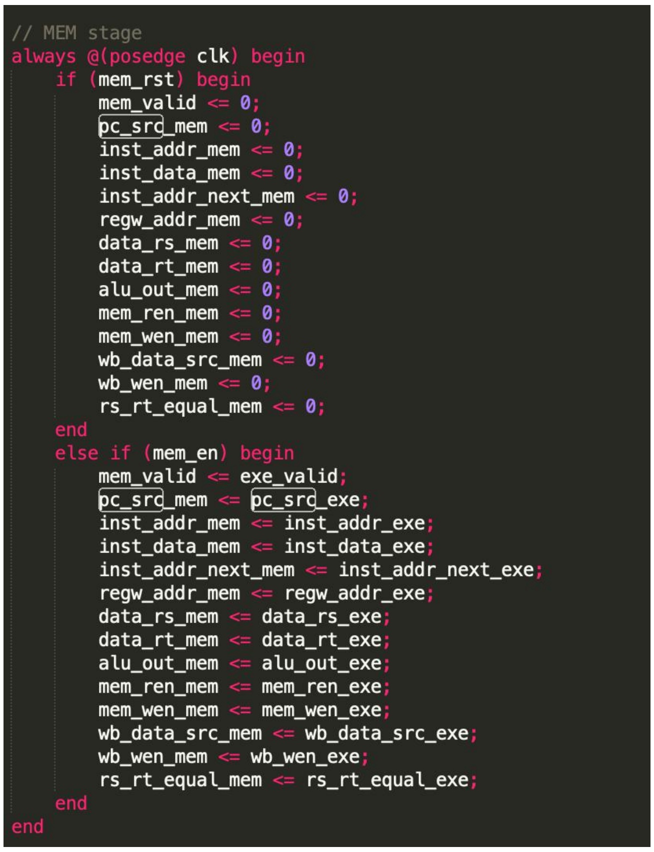
```
// ID stage
always @(posedge clk) begin
    if (id_rst) begin
        id_valid <= 0;
        inst_addr_id <= 0;
        inst_data_id <= 0;
        inst_addr_next_id <= 0;
    end
    else if (id_en) begin
        id_valid <= if_valid;
        inst_addr_id <= inst_addr;
        inst_data_id <= inst_data;
        inst_addr_next_id <= inst_addr_next;
    end
end
```

EXE stage with rst signal:



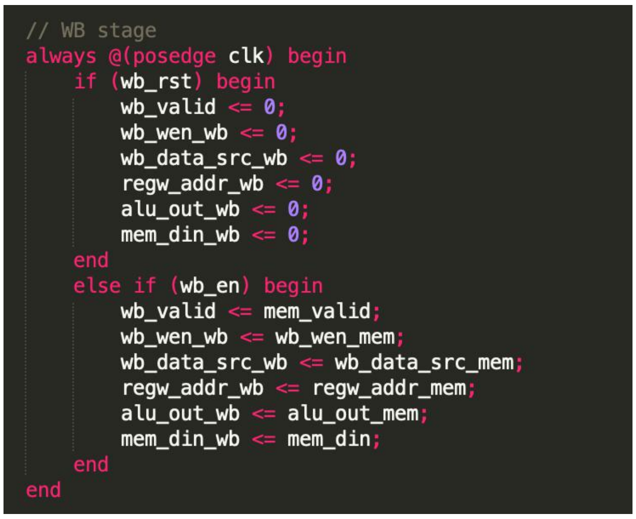
```
// EXE stage
always @(posedge clk) begin
    if (exe_rst) begin
        exe_valid <= 0;
        inst_addr_exe <= 0;
        inst_data_exe <= 0;
        inst_addr_next_exe <= 0;
        regw_addr_exe <= 0;
        pc_src_exe <= 0;
        exe_a_src_exe <= 0;
        exe_b_src_exe <= 0;
        data_rs_exe <= 0;
        data_rt_exe <= 0;
        data_imm_exe <= 0;
        exe_alu_oper_exe <= 0;
        mem_ren_exe <= 0;
        mem_wen_exe <= 0;
        wb_data_src_exe <= 0;
        wb_wen_exe <= 0;
    end
    else if (exe_en) begin
        exe_valid <= id_valid;
        inst_addr_exe <= inst_addr_id;
        inst_data_exe <= inst_data_id;
        inst_addr_next_exe <= inst_addr_next_id;
        regw_addr_exe <= regw_addr_id;
        pc_src_exe <= pc_src_ctrl;
        exe_a_src_exe <= exe_a_src_ctrl;
        exe_b_src_exe <= exe_b_src_ctrl;
        data_rs_exe <= data_rs;
        data_rt_exe <= data_rt;
        data_imm_exe <= data_imm;
        exe_alu_oper_exe <= exe_alu_oper_ctrl;
        mem_ren_exe <= mem_ren_ctrl;
        mem_wen_exe <= mem_wen_ctrl;
        wb_data_src_exe <= wb_data_src_ctrl;
        wb_wen_exe <= wb_wen_ctrl;
    end
end
```

MEM stage with rst signal:



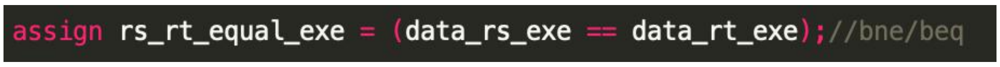
```
// MEM stage
always @(posedge clk) begin
    if (mem_rst) begin
        mem_valid <= 0;
        pc_src_mem <= 0;
        inst_addr_mem <= 0;
        inst_data_mem <= 0;
        inst_addr_next_mem <= 0;
        regw_addr_mem <= 0;
        data_rs_mem <= 0;
        data_rt_mem <= 0;
        alu_out_mem <= 0;
        mem_ren_mem <= 0;
        mem_wen_mem <= 0;
        wb_data_src_mem <= 0;
        wb_wen_mem <= 0;
        rs_rt_equal_mem <= 0;
    end
    else if (mem_en) begin
        mem_valid <= exe_valid;
        pc_src_mem <= pc_src_exe;
        inst_addr_mem <= inst_addr_exe;
        inst_data_mem <= inst_data_exe;
        inst_addr_next_mem <= inst_addr_next_exe;
        regw_addr_mem <= regw_addr_exe;
        data_rs_mem <= data_rs_exe;
        data_rt_mem <= data_rt_exe;
        alu_out_mem <= alu_out_exe;
        mem_ren_mem <= mem_ren_exe;
        mem_wen_mem <= mem_wen_exe;
        wb_data_src_mem <= wb_data_src_exe;
        wb_wen_mem <= wb_wen_exe;
        rs_rt_equal_mem <= rs_rt_equal_exe;
    end
end
```

WB stage:



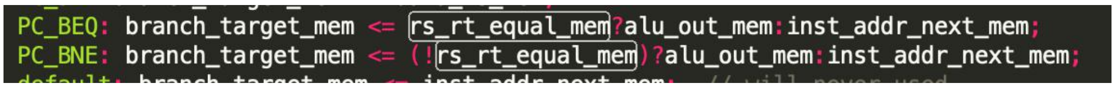
```
// WB stage
always @(posedge clk) begin
    if (wb_rst) begin
        wb_valid <= 0;
        wb_wen_wb <= 0;
        wb_data_src_wb <= 0;
        regw_addr_wb <= 0;
        alu_out_wb <= 0;
        mem_din_wb <= 0;
    end
    else if (wb_en) begin
        wb_valid <= mem_valid;
        wb_wen_wb <= wb_wen_mem;
        wb_data_src_wb <= wb_data_src_mem;
        regw_addr_wb <= regw_addr_mem;
        alu_out_wb <= alu_out_mem;
        mem_din_wb <= mem_din;
    end
end
```

3.5.detail in the bne and beq instruction:



```
assign rs_rt_equal_exe = (data_rs_exe == data_rt_exe);//bne/beq
```

It judge in the mem stage:

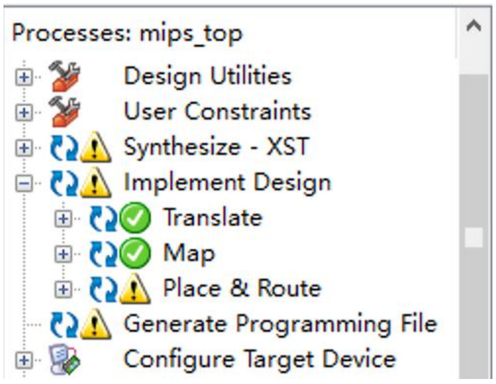


```
PC_BEQ: branch_target_mem <= rs_rt_equal_mem?alu_out_mem:inst_addr_next_mem;
PC_BNE: branch_target_mem <= (!rs_rt_equal_mem)?alu_out_mem:inst_addr_next_mem;
```

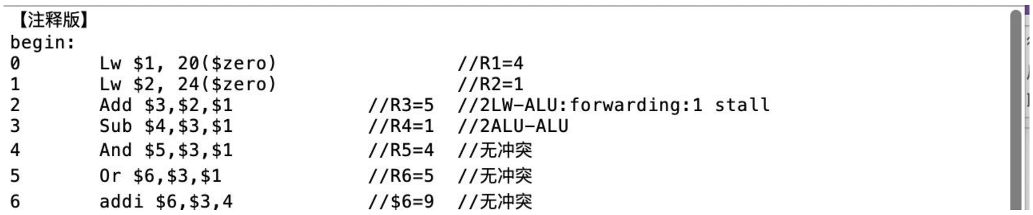
If it is not taken, then is_branch_mem will be 0, and a bubble is saved, so it is to predict taken. But it still waste a lot, and I will put the judge in the ID stage in my forwarding job.

四、实验结果分析

编译 bit 文件:

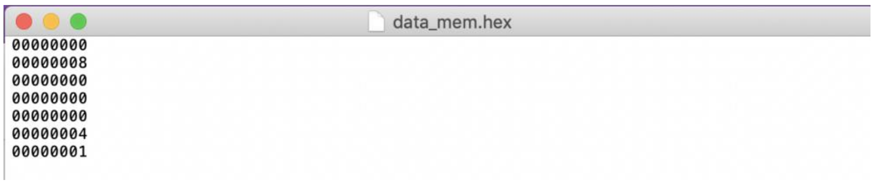


Use the code given:



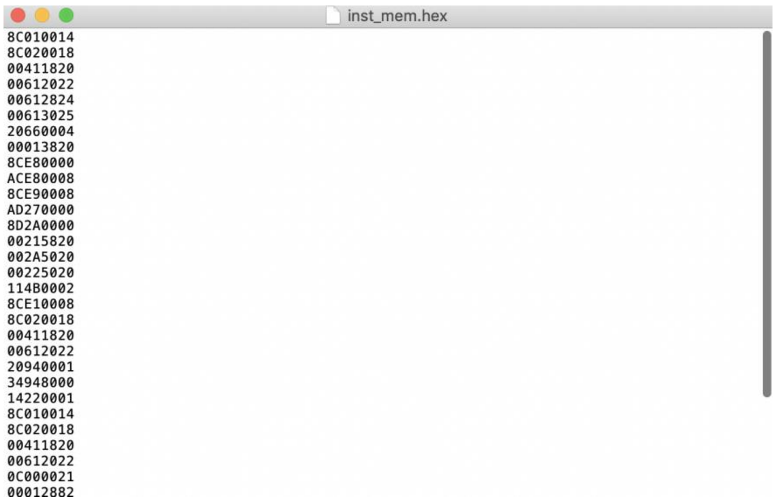
```
【注释版】
begin:
0       Lw $1, 20($zero)                //R1=4
1       Lw $2, 24($zero)                //R2=1
2       Add $3,$2,$1            //R3=5  //2LW-ALU:forwarding:1 stall
3       Sub $4,$3,$1            //R4=1  //2ALU-ALU
4       And $5,$3,$1            //R5=4  //无冲突
5       Or $6,$3,$1    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

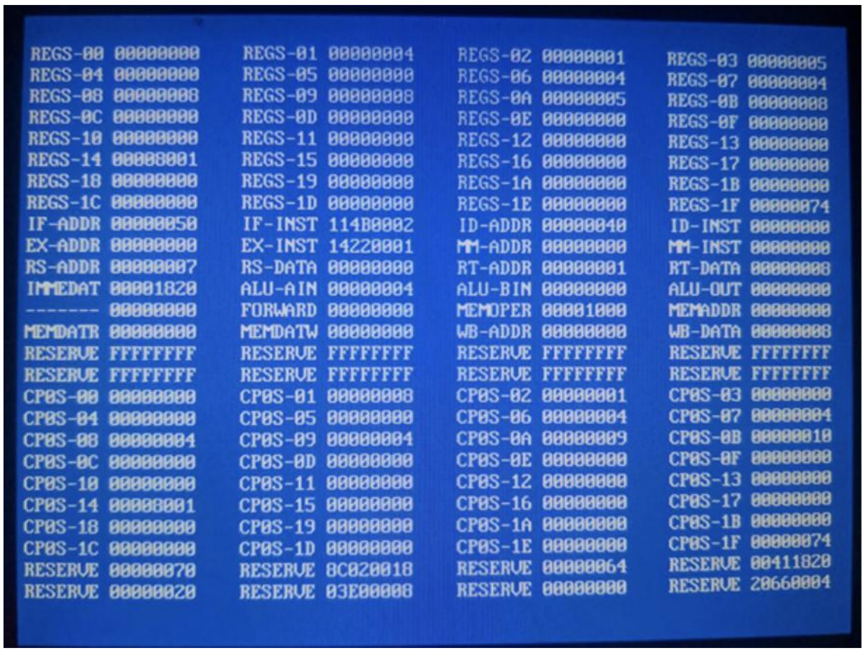
```
8C010014
8C020018
00411820
00612022
00612824
00613025
20660004
00013820
8CE80000
ACE80008
8CE90008
AD270000
8D2A0000
00215820
002A5020
00225020
114B0002
8CE10008
8C020018
00411820
00612022
20940001
34948000
14220001
8C010014
8C020018
00411820
00612022
0C000021
00012882
```

Vga:

When it comes to, because there is no forwarding it will insert two bubbles:

```
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

6        addi $6,$3,4           //$6=9  //无冲突
7        Add $7, $zero, $1              //R7=4  //无冲突
8        Lw $8,0($7)            //R8=8  //2ALU-LW
9        Sw $8,8($7)            //      //2LW-SW: forwarding可以解决
```

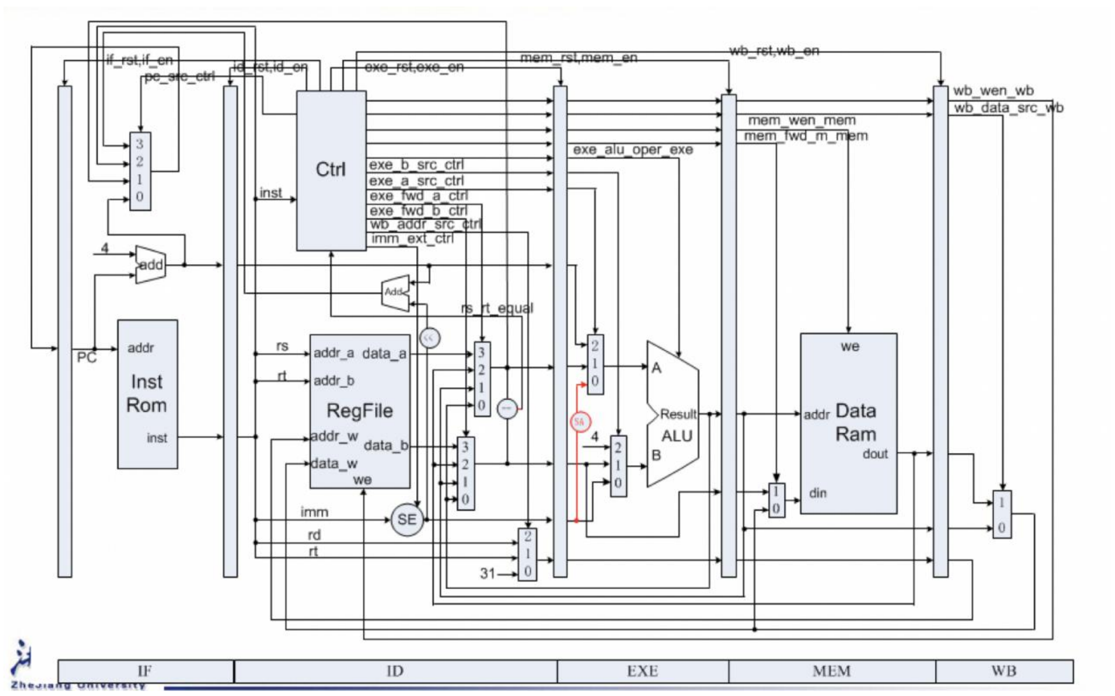
And the result is the same as what we predict, so the stall is working!



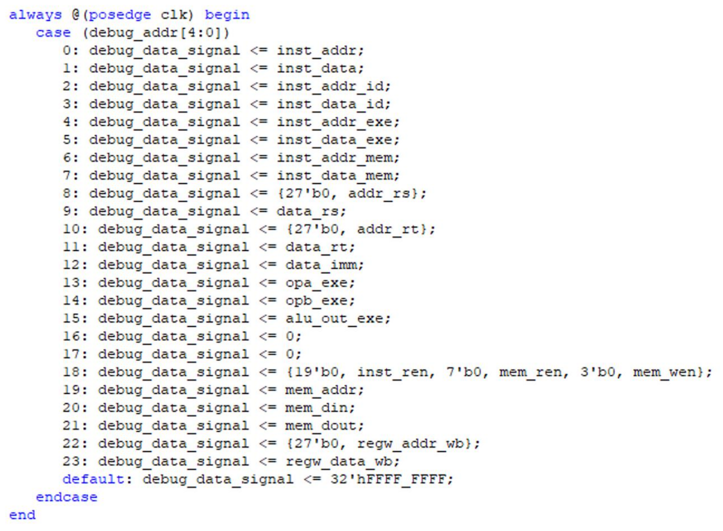
```
REGS-00  00000000   REGS-01  00000004   REGS-02  00000001   REGS-03  00000005
REGS-04  00000000   REGS-05  00000000   REGS-06  00000004   REGS-07  00000004
REGS-08  00000008   REGS-09  00000008   REGS-0A  00000005   REGS-0B  00000008
REGS-0C  00000000   REGS-0D  00000000   REGS-0E  00000000   REGS-0F  00000000
REGS-10  00000000   REGS-11  00000000   REGS-12  00000000   REGS-13  00000000
REGS-14  00000001   REGS-15  00000000   REGS-16  00000000   REGS-17  00000000
REGS-18  00000000   REGS-19  00000000   REGS-1A  00000000   REGS-1B  00000000
REGS-1C  00000000   REGS-1D  00000000   REGS-1E  00000000   REGS-1F  00000074
IF-ADDR  00000050   IF-INST  114B0002   ID-ADDR  00000040   ID-INST  00000000
EX-ADDR  00000000   EX-INST  14220001   MM-ADDR  00000000   MM-INST  00000000
RS-ADDR  00000007   RS-DATA  00000000   RT-ADDR  00000001   RT-DATA  00000008
IMMEDAT  00001820   ALU-AIN  00000004   ALU-BIN  00000000   ALU-OUT  00000000
-------  00000000   FORWARD  00000000   MEMOPER  00001000   MEMADDR  00000000
MEMDATR  00000000   MEMDATW  00000000   WB-ADDR  00000000   WB-DATA  00000008
RESERVE  FFFFFFFF   RESERVE  FFFFFFFF   RESERVE  FFFFFFFF   RESERVE  FFFFFFFF
RESERVE  FFFFFFFF   RESERVE  FFFFFFFF   RESERVE  FFFFFFFF   RESERVE  FFFFFFFF
CP0S-00  00000000   CP0S-01  00000008   CP0S-02  00000001   CP0S-03  00000000
CP0S-04  00000000   CP0S-05  00000000   CP0S-06  00000004   CP0S-07  00000004
CP0S-08  00000004   CP0S-09  00000004   CP0S-0A  00000009   CP0S-0B  00000010
CP0S-0C  00000000   CP0S-0D  00000000   CP0S-0E  00000000   CP0S-0F  00000000
CP0S-10  00000000   CP0S-11  00000000   CP0S-12  00000000   CP0S-13  00000000
CP0S-14  00000001   CP0S-15  00000000   CP0S-16  00000000   CP0S-17  00000000
CP0S-18  00000000   CP0S-19  00000000   CP0S-1A  00000000   CP0S-1B  00000000
CP0S-1C  00000000   CP0S-1D  00000000   CP0S-1E  00000000   CP0S-1F  00000074
RESERVE  00000070   RESERVE  8C020018   RESERVE  00000064   RESERVE  00411820
RESERVE  00000020   RESERVE  03E00008   RESERVE  00000000   RESERVE  20660004
```

五、实验总结与反思

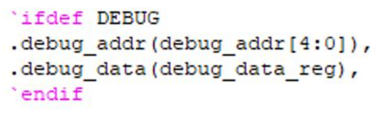
1.Specifically, in the implementation of the bne code, the calculation of the address in the EXE stage and the judgment of whether to jump are selected in this stage. In order to facilitate future experiments, the steps of judgment and address calculation should be moved forward. Moreover, in order to advance the ALU module used to calculate the address, an adder should be added as the followings: **so the next one I will use this datapath**



2.The vga debugger is need to be added into every module that has the signals in the following code, and be careful not to miss any:


```
always @(posedge clk) begin
   case (debug_addr[4:0])
      0: debug_data_signal <= inst_addr;
      1: debug_data_signal <= inst_data;
      2: debug_data_signal <= inst_addr_id;
      3: debug_data_signal <= inst_data_id;
      4: debug_data_signal <= inst_addr_exe;
      5: debug_data_signal <= inst_data_exe;
      6: debug_data_signal <= inst_addr_mem;
      7: debug_data_signal <= inst_data_mem;
      8: debug_data_signal <= {27'b0, addr_rs};
      9: debug_data_signal <= data_rs;
      10: debug_data_signal <= {27'b0, addr_rt};
      11: debug_data_signal <= data_rt;
      12: debug_data_signal <= data_imm;
      13: debug_data_signal <= opa_exe;
      14: debug_data_signal <= opb_exe;
      15: debug_data_signal <= alu_out_exe;
      16: debug_data_signal <= 0;
      17: debug_data_signal <= 0;
      18: debug_data_signal <= {19'b0, inst_ren, 7'b0, mem_ren, 3'b0, mem_wen};
      19: debug_data_signal <= mem_addr;
      20: debug_data_signal <= mem_din;
      21: debug_data_signal <= mem_dout;
      22: debug_data_signal <= {27'b0, regw_addr_wb};
      23: debug_data_signal <= regw_data_wb;
      default: debug_data_signal <= 32'hFFFF_FFFF;
   endcase
end
```

For example in the Reg:



```
`ifdef DEBUG
.debug_addr(debug_addr[4:0]),
.debug_data(debug_data_reg),
`endif
```