实验项目名称：Pipelined CPU with stall

一、 实验目的和要求

1. Design the Stall Part of Datapath of 5-stages Pipelined CPU
2. Modify the CPU Controller, adding Condition Detection of Stall.
3. Verify the Pipelined CPU with program and observe the execution of program

二、 实验内容和原理

1. Data Hazard

Data hazards occur when the pipeline changes the order of read/write accesses to operands comparing with that in sequential executing and there exits an instruction that depends on the results of a previous instruction:

1) RAW: when the instruction reads a source before another instruction writes it.
2) WAW: when the instruction writes an operand before another instruction writes it.
3) WAR: when the instruction writes a destination before another instruction reads it.

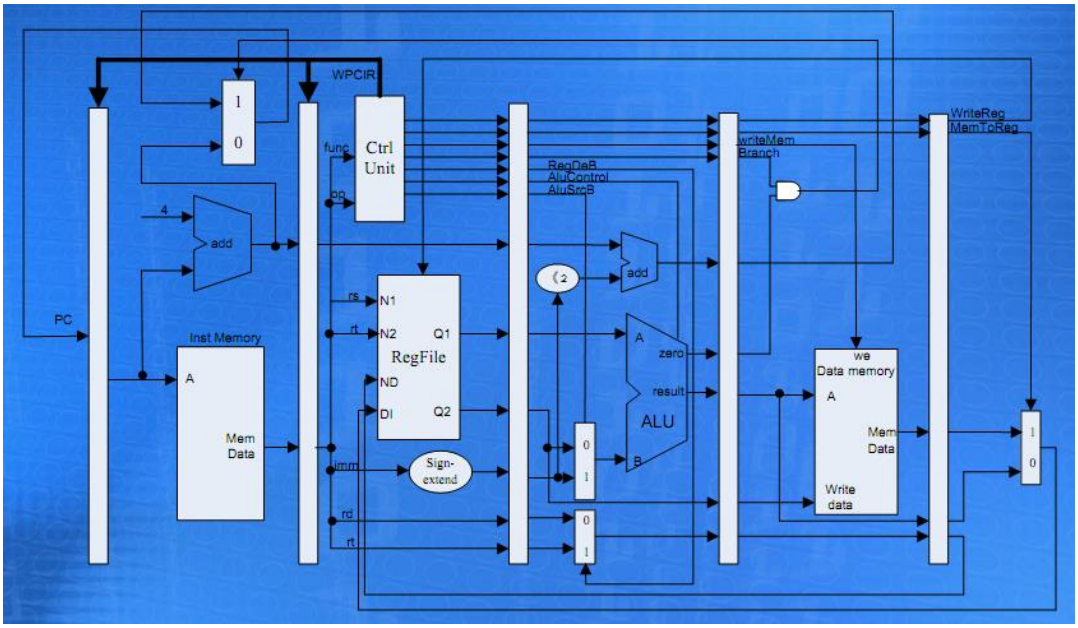
2. Stalls to resolve data hazards

The solution is to detect stall situations at ID stage and insert stalls until there are no data hazards

1) Data hazard detection: Check whether the source(rs or rt) of an instruction in ID stage is the same as the destination(rd for ALU instructions or rt for LW) of another instruction in EX or MEM stage.

   Note that we've used double bump, so instructions in WB stage can be ignored. And the detection is implemented in ctrl_unit module.

2) The datapath of 5-stages Pipelined CPU with stalls is as follows:



三、 实验过程和数据记录

The modified code of the ctrl_unit module:

```
module ctrl_unit(clk, rst, if_instr, instr, cu_branch, cu_wreg, cu_m2reg,
cu_wmem, cu_aluc, cu_shift, cu_aluimm, cu_sext, cu_regrt, cu_wpcir);
    output cu_wpcir;
    //stall signals
    wire AfromEx;
    wire BfromEx;
    wire AfromMem;
    wire BfromMem;
    wire AfromExLW;
    wire BfromExLW;
    wire AfromMemLW;
    wire BfromMemLW;
    wire stall;
    wire cu_wpcir;

    assign AfromEx = (if_rs==rd) & (if_rs!=0) & (opcode==`OP_ALUOp);
    assign BfromEx = (if_rt==rd) & (if_rt!=0) & (opcode==`OP_ALUOp);
    assign AfromMem = (if_rs==ex_rd) & (if_rs!=0) & (ex_op==`OP_ALUOp);
    assign BfromMem = (if_rt==ex_rd) & (if_rt!=0) & (ex_op==`OP_ALUOp);
    assign AfromExLW = (if_rs==rt) & (if_rs!=0) & (opcode==`OP_LW);
    assign BfromExLW = (if_rt==rt) & (if_rt!=0) & (opcode==`OP_LW);
    assign AfromMemLW = (if_rs==ex_rt) & (if_rs!=0) & (ex_op==`OP_LW);
    assign BfromMemLW = (if_rt==ex_rt) & (if_rt!=0) & (ex_op==`OP_LW);
    assign stall = AfromEx || BfromEx || AfromMem || BfromMem || AfromExLW
|| BfromExLW || AfromMemLW || BfromMemLW;
    assign cu_wpcir = stall;
```

The modified code of the id_stage module:

```
module id_stage (clk, rst, if_inst, if_pc4, wb_destR, wb_dest,wb_wreg,
cu_wreg, cu_m2reg, cu_wmem, cu_aluc, cu_shift, cu_aluimm, cu_branch,
id_pc4, id_inA, id_inB, id_imm, cu_regrt, rt, rd, cu_wpcir, IF_ins_type,
IF_ins_number, ID_ins_type, ID_ins_number, which_reg, reg_content);
    output cu_wpcir;  //newly added output
    wire cu_wpcir;
    always @ (posedge clk or posedge rst)
        if (rst==1)
```

```
        begin
            pc4 <= 32'hffffffff;
            ID_ins_type <= 4'b0000;
            ID_ins_number <= 4'b0000;
        end
        else
        begin
//when without stalls, renew reg_inst,pc4,ID_ins_number and ID_ins_type
            if(cu_wpcir == 0)
            begin
                reg_inst <= if_inst;
                pc4 <= if_pc4;
                ID_ins_type <= IF_ins_type;
                ID_ins_number <= IF_ins_number;
            end
            else   //when there exit stalls
            begin
                reg_inst <= 0;    //set reg_inst to zero
                //ID_ins_number and pc4 will not be changed
                //set ID_ins_type to ID_ins_type
                ID_ins_type <= `INST_TYPE_NONE;
            end
        end
    //get the cu_wpcir signal from ctrl_unit module
    ctrl_unit x_ctrl_unit(clk, rst, if_inst[31:0], reg_inst[31:0],
    cu_branch, cu_wreg, cu_m2reg, cu_wmem, cu_aluc, cu_shift, cu_aluimm,
cu_sext, cu_regrt, cu_wpcir);
```

The modified code of the if_stage module:

```
module if_stage (mem_clk, clk, rst, npc, nid_pc, ctrl_branch, id_wpcir,
if_pc,  if_pc4,  if_inst,  IF_ins_type,  IF_ins_number,  ID_ins_type,
ID_ins_number);
    input id_wpcir;   //newly added input
    wire id_wpcir;
    always @ (posedge clk) begin
        if(id_wpcir == 0) begin//renew pc and set run when without stalls
            pc[31:0] <= npc[31:0];
            run <= 1'b1;
        end
        //when there're stalls, pc will not be change
    end
    always @(if_inst) begin
        if (run == 1'b0) begin   //for initial 0
            ID_ins_type[3:0] <= 4'b0000;
            ID_ins_number[3:0] <= 4'b0000;
        end
        else
        begin
            //when there're no stalls, renew ID_ins_number and ID_ins_type
            if(id_wpcir == 0) begin
                ID_ins_number[3:0] <= pc[3:0];
                //omit some codes here
            end
//when there're stalls, ID_ins_number and ID_ins_type will not be changed
        end
    end
```

The modified code of the top module:

```
    reg [31:0] if_inst_old;     //store the formal value of if_inst
    reg [31:0] reg_content_old; //store the formal value of reg_content
    initial begin //initialize the old values to zero
```

```
       if_inst_old <= 32'b0;
       reg_content_old <= 32'b0;
    end
    always @(posedge CCLK  ) begin
       //real-time display the instruction
       if ((BTN3 == 1'b1) || (BTN2 == 1'b1)||(if_inst_old != if_inst))
begin
          if_inst_old <= if_inst;
          //first line 8 4-bit Instrution
          if (if_inst[31:28] < 10)
             strdata[255:248] <= 8'h30 + if_inst[31:28];
          else
             strdata[255:248] <= 8'h61 + if_inst[31:28] - 10;
          //omit some codes here
       //real-time display the content of register
       if((BTN3 == 1'b1) || (BTN2 == 1'b1)||(SW_old != SW)|| (BTN0 !=
BTN_OLD) || (reg_content_old != reg_content))
       begin
          BTN_OLD <= BTN0;
          //first line after CLK and space
          if (BTN0 == 1'b1)  begin
             if (reg_content[15:12] < 10)  //upper 16-bit
                strdata[159:152] <= 8'h30 + reg_content[31:28];
             else
                strdata[159:152] <= 8'h61 + reg_content[31:28] - 10;
             //omit some codes here
          end
          else begin
             //omit some codes here
          end
          reg_content_old<= reg_content;
          SW_old <= SW;
          cls <= 1;  //set the flag
       end
       else begin
         cls <= 0;    //reset the flag
       end
     end
    //pass id_wpcir from id_stage to if_stage module
   if_stage x_if_stage(clk0, BTN3, rst, pc, mem_pc, mem_branch, id_wpcir,
if_npc,  if_pc4,  if_inst,  IF_ins_type,  IF_ins_number,  ID_ins_type,
ID_ins_number);

    id_stage x_id_stage(BTN3, rst, if_inst, if_pc4, wb_destR, wb_dest,
wb_wreg,  id_wreg,  id_m2reg,  id_wmem,  id_aluc,  id_shift,  id_aluimm,
id_branch,  id_pc4,  id_inA,  id_inB,  id_imm,  id_regrt,  id_rt,  id_rd,
id_wpcir,  ID_ins_type,  ID_ins_number,  EX_ins_type,  EX_ins_number,
{1'b0,which_reg}, reg_content);
```

四、 实验结果分析

Program for verification is as follows:

MIPS code	Bin data	Address	Inst. Type
lw r1, $20(r0)	0x8c01_0014	0	6
lw r2, $21(r0)	0x8c02_0015	1	6
add r3, r1, r2	0x0022_1820	2	1

add r2,r0,r0	0x0000_1020	3	1
sub r4, r1, r3	0x0023_2022	4	2
and r5, r3, r4	0x0064_2824	5	3
nor r6, r4, r5	0x0085_3027	6	5
sw r6, $22(r0)	0xac06_0016	7	7
beq r6,r7,-8	0x10c7_fff8	8	8

Data Mem[20]=1

Data Mem[21]=4

The result of execution is shown as follows:

Clock Count	Instruction Code	Instruction	“stage name” /number/type	Comment
00	01234567		f01d01e01m01w01	The first instruction “lw r1, $20(r0)” enters the IF stage.
01	8c010014	lw r1, $20(r0)	f1fd06e00m00w00	The first instruction “lw r1, $20(r0)” enters the ID stage. And a new instruction “lw r2, $21(r0)” enters the IF stage.
02	8c020015	lw r2, $21(r0)	f2fd16e06m00w00	The first instruction “lw r1, $20(r0)” enters the EX stage. And a new instruction “ add r3, r1, r2” enters the IF stage.
03	00221820	add r3, r1, r2	f3fd26e16m06w00	The first instruction “lw r1, $20(r0)” enters the MEM stage. And

				the instruction “add r3, r1, r2” enters the ID stage and find that there exists a data hazard so a stall is inserted .
04	00221820	add r3, r1, r2	f3fd26e1fm16w06	The first instruction “lw r1, $20(r0)” enters the WB stage.
05	00221820	add r3, r1, r2	f3fd21e1fm1fw16	The first instruction “lw r1, $20(r0)” completes the WB stage, so the content of r1 becomes 1.
06	00001020	add r2,r0,r0	f4fd31e21m1fw1f	The instruction “lw r2, $21(r0)” completes the WB stage, so the content of r2 becomes 4.
07	00232022	sub r4, r1, r3	f5fd42e31m21w1f	The stall completes the WB stage.
08	00232022	sub r4, r1, r3	f5fd42e3fm31w21	The stall completes the WB stage.
09	00642824	and r5, r3, r4	f6fd53e42m3fw31	The instruction “add r3, r1, r2” completes the WB stage, so the

				content of r3 becomes 5.
0a	00642824	and r5, r3, r4	f6fd53e4fm42w3f	The instruction “add r2, r0, r0” completes the WB stage, so the content of r3 becomes 0.
0b	00642824	and r5, r3, r4	f6fd53e4fm4fw42	The stall completes the WB stage.
0c	00853027	nor r6, r4, r5	f7fd65e53m4fw4f	The instruction “sub r4, r1, r3” completes the WB stage, so the content of r4 becomes 0x ffff fffc.
0d	00853027	nor r6, r4, r5	f7fd65e5fm53w4f	The stall completes the WB stage.
0e	00853027	nor r6, r4, r5	f7fd65e5fm5fw53	The stall completes the WB stage.
0f	ac060016	sw r6, $22(r0)	f8fd77e65m5fw5f	The instruction “and r5, r3, r4” completes the WB stage, so the content of r5 becomes 4.
10	ac060016	sw r6, $22(r0)	f8fd77e6fm65w5f	The stall completes the WB stage.

11	ac060016	sw r6, $22(r0)	f8fd77e6fm6fw65	The stall completes the WB stage.
12	10c7fff8	beq r6,r7,-8	f9fd88e77m6fw6f	The instruction “nor r6, r4, r5” completes the WB stage, so the content of r6 becomes 3.
13	00000000	nop	fafd97e88m77w6f	The stall completes the WB stage.
14	00000000	nop	fbfdace9cm88w77	The stall completes the WB stage.
15	00000000	nop	fcfdbceacm9cw88	The instruction “sw r6, $22(r0)” completes the WB stage.
16	00000000	nop	fdfdccebcmacw9c	The instruction “beq r6,r7,-8” completes the WB stage.

The result indicates that the code implements the required functions correctly.

五、 讨论与心得

In the result of the previous lab “5-stage CPU executing in pipeline”, we can find that the display of instruction type of ID stage(ID_ins_type) is not in sync with that of the ID_ins_number. The reason is that according to the code of the if_stage module, ID_ins_type is changed with the if_inst, which is not real-time displayed. So when we add real-time display in top module, the problem above is solved.