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

Lab5—Pipelined CPU with forwarding

课程名称：　计算机体系结构　　实验类型：　综合

实验项目名称：　用 forwarding 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 Bypass Unit

•Master the method of Pipelined Pipeline Forwarding Detection and Pipeline Forwards.

•Master the Condition In Which Pipeline Forwards.

•Master the Condition In Which Bypass Unit doesn't Work and the Pipeline stalls.

•master methods of program verification of Pipelined CPU with forwarding

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 Bypass Unit of Datapath of 5-stages Pipelined CPU

•Modify the CPU Controller

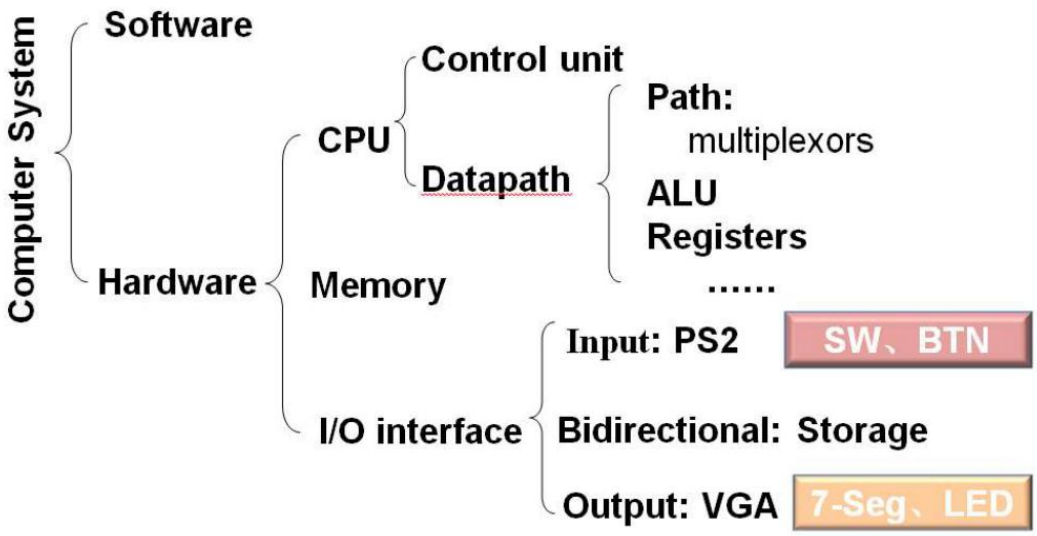
–Conditions in Which Pipeline Forwards.

–Conditions in Which Pipeline Stalls.

•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.Data Hazard Stalls

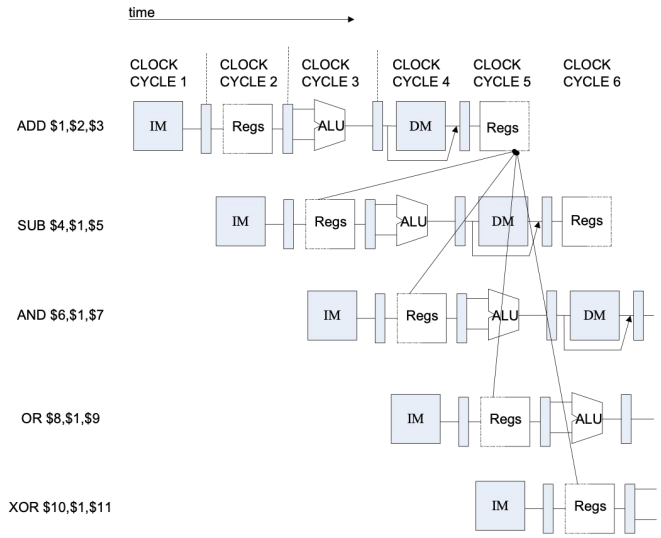
•Minimizing Data Hazard Stalls by Forwarding:

In most cases, the problem can be resolved by forwarding, also called bypassing, short-circuiting.

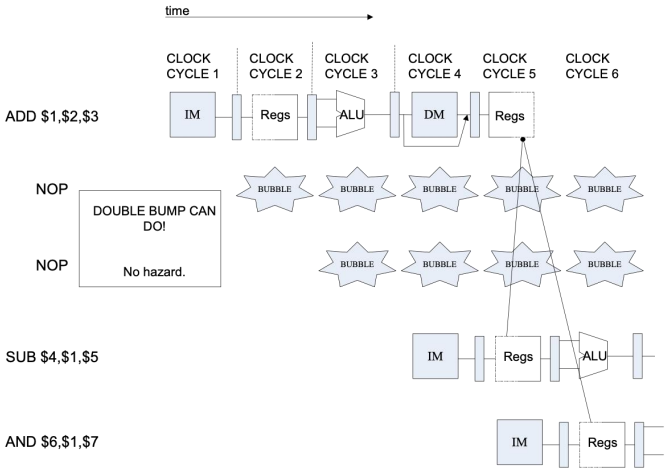
•Data Hazards Requiring Stalls:

In some cases, data hazards can not be handled by bypassing.

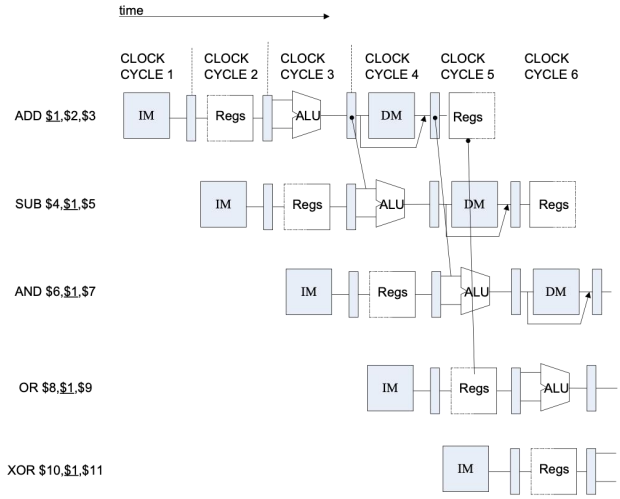
Instruction demo:



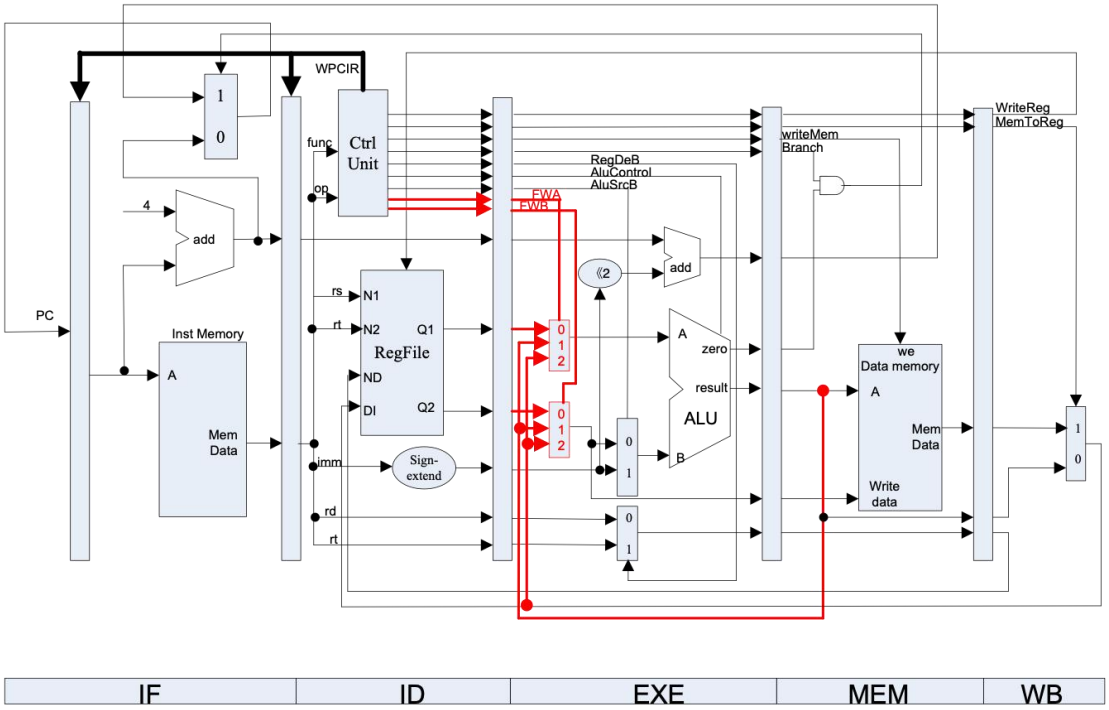
Data Hazard Causes Stalls:



Pipeline Forward to Avoid the Data hazard:

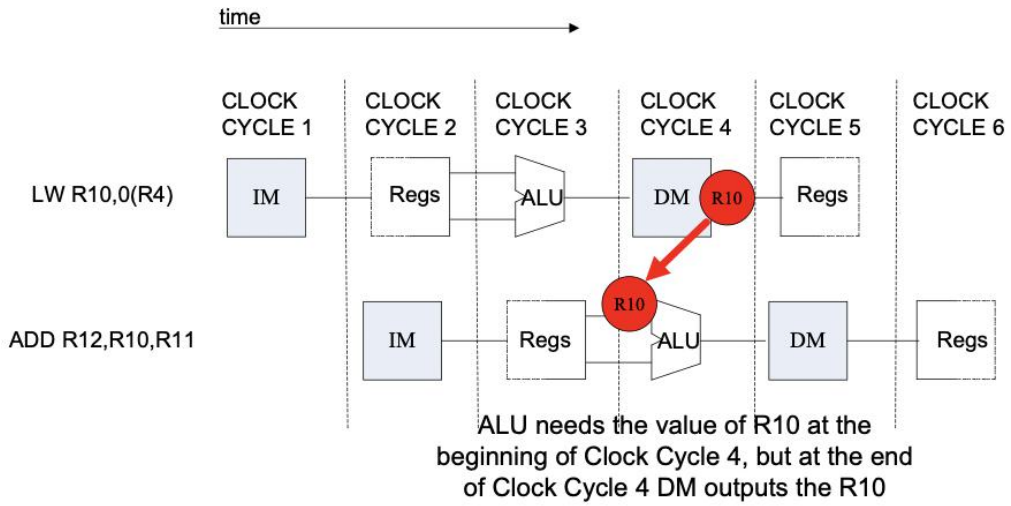


Data path with Bypass Unit



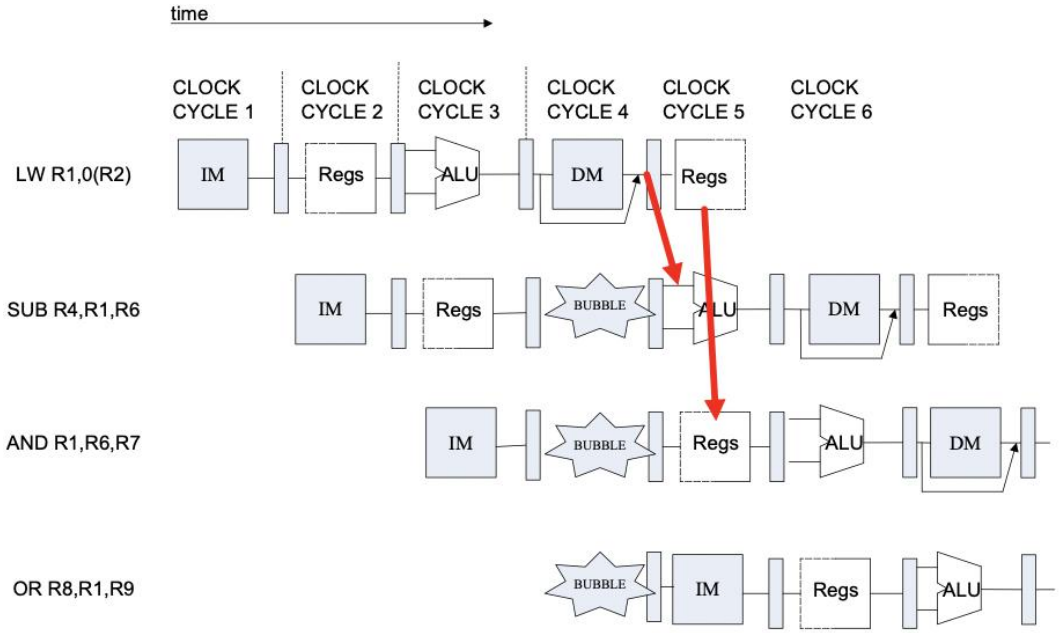
2.2.3.Conditions in which bypass unit doesn't work

First condition:



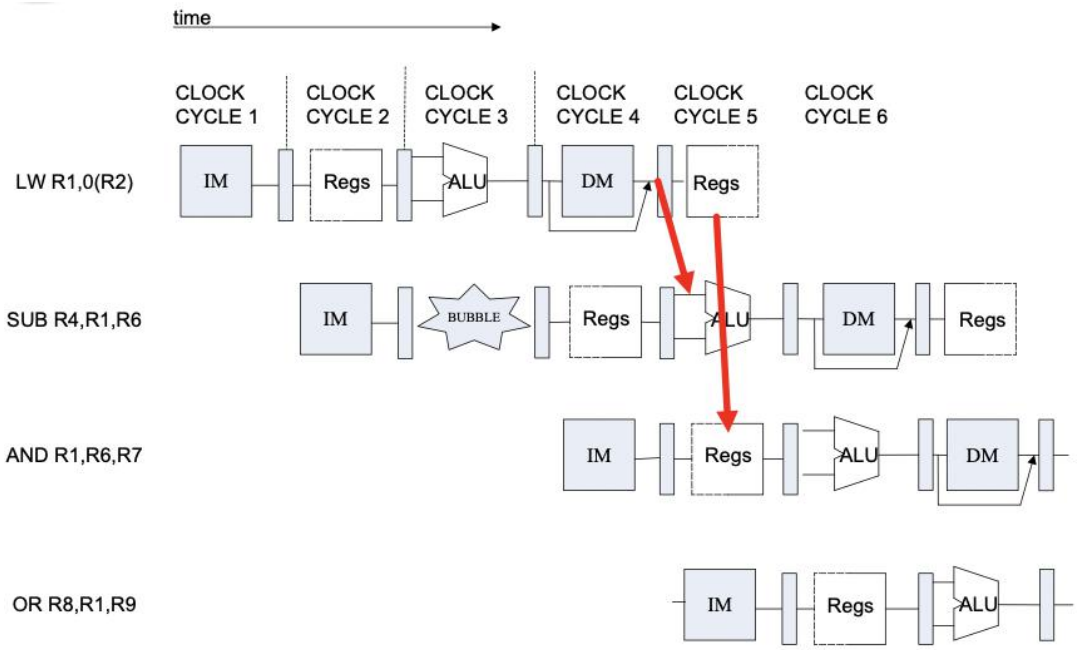
Second condition:

Pipeline Stalls



Third condition:

Pipeline Stalls at ID Stage



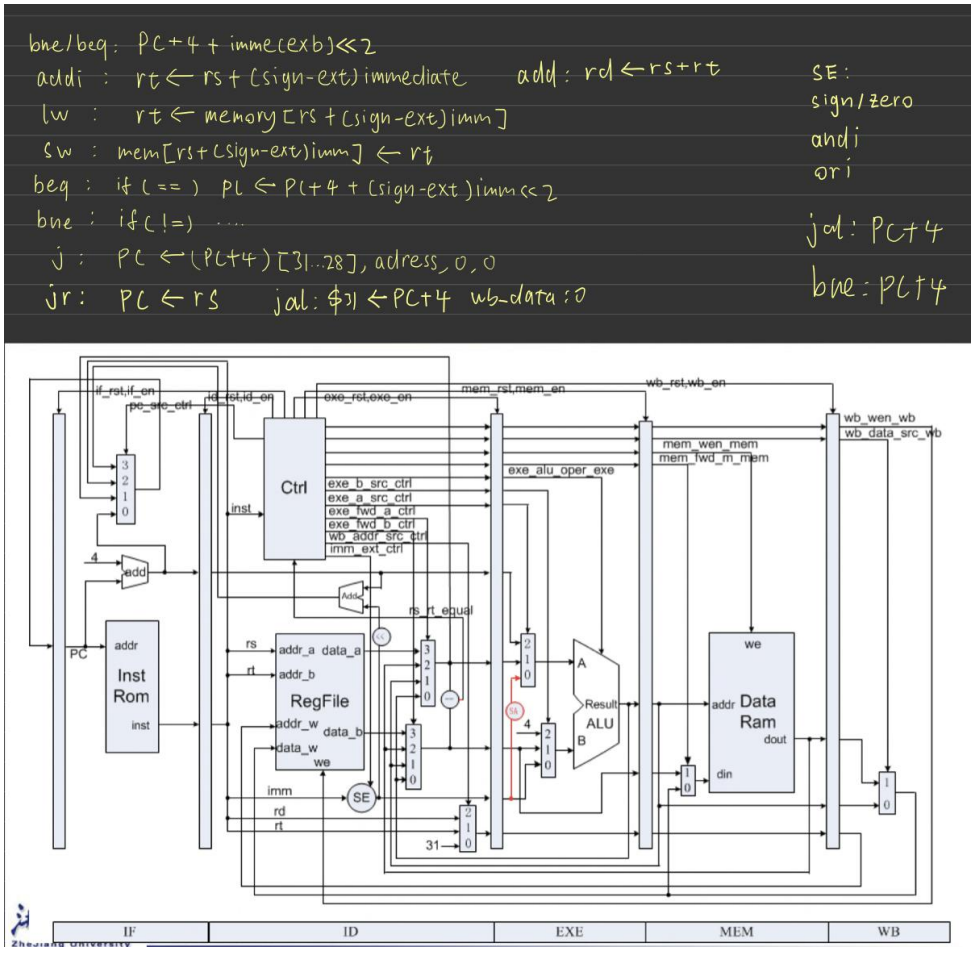
- **Condition in which pipeline forwards and stalls**

```
…
assign AfromEx = …              //ALUOp
assign BfromEx = …              //ALUOp
assign AfromMem = …             //ALUOp + LW
assign BfromMem = …             //ALUOp + LW
assign AfromExLW = …            //与 if_inst 比
assign BfromExLW = …            //与 if_inst 比
…
assign cu_fwda[1:0] = (AfromEx == 1)? 2'b01 :((AfromMem == 1)?2'b10:2'b00);
assign cu_fwdb[1:0] = (BfromEx == 1)? 2'b01 :((BfromMem == 1)?2'b10:2'b00);
…
assign stall = AfromExLW || BfromExLW;
…
```

one more condition：LW+SW

三、实验过程

3.1.Design the forwarding data path: 提前了判断和地址拼接



3.2.To detect the condition：

There are four possible choices about the fwa and fwb data from different sources:

--0: read alu result in exe stage;

--1: read alu result in mem stage;

--2: read dout in mem stage;

--3: normal ones

- **If read rs next to LW, we need a load stall and choose the dout**

```
    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;
```

- **If read rt next to LW, two possible situations:**

SW next to LW, no EX phase:

other instructions next to LW, have EX phase:

```
else if(rt_used && regw_addr_exe == addr_rt) begin
    // read rt next to LW, 2 possible situations
    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
```

- **data read from memory(after 1 cycle stall)**

```
 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
```

3.3.Load stall

To insert NOPs between ID and EXE.

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

3.4.In each stage, there are some changes:

ID:

```
// new forwarding
always @(*) begin
    data_rs_id = data_rs;
    data_rt_id = data_rt;
    case(exe_fwd_a_ctrl)
        0: data_rs_id = alu_out_exe;
        1: data_rs_id = alu_out_mem;
        2: data_rs_id = mem_din;
    endcase
    case(exe_fwd_b_ctrl)
        0: data_rt_id = alu_out_exe;
        1: data_rt_id = alu_out_mem;
        2: data_rt_id = mem_din;
    endcase
end
```

EXE:

```
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;

    // new forwarding
    pc_src_exe <= pc_src_ctrl;
    data_rs_exe <= data_rs_id;
    data_rt_exe <= data_rt_id;
    is_load_exe <= is_load_ctrl;
    is_store_exe <= is_store_ctrl;
    fwd_m_exe <= fwd_m_ctrl;

    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
```

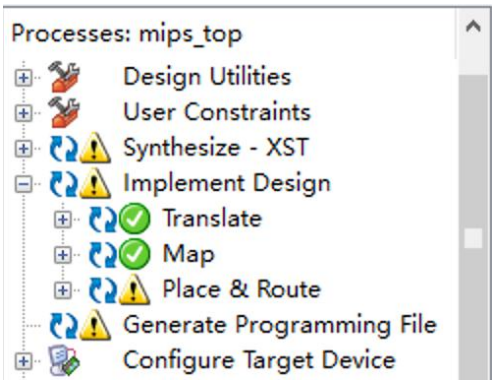
IN MEM:

```
// 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;

        // new forwarding
        fwd_m_mem <= 0;
        is_load_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;

        // new forwarding
        fwd_m_mem <= fwd_m_exe;
        is_load_mem <= is_load_exe;
    end
end
```

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

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 a forwarding it will insert only one bubble:

```
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
```

```
a       Lw $9, 8($7)            //R9=8
b       Sw $7, 0($9)            //      //2LW-SW: forwarding:1 stall
c       Lw $10,0($9)            //R10=4
d       Add $11, $1, $1         //R11=8
e       Add $10, $1, $10        //R10=8 //1LW-ALU:forwarding可以解决
f       Add $10,$1,$2           //R10=5
10      Beq $10, $11 ,lable1    // not taken//2+1ALU-BEQ//      branch x
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
18      Lw $1,20($zero)         //不执行
lable2:
19      Lw $2,24($zero)         //R2=1
1a      Add $3,$2,$1            //R3=9   //2ALU
1b      Sub $4,$3,$1            //R4=1   //2ALU x
1c      Jal lable3                      //R31=116(0x74)
1d      Srl $5,$1,2             //R5=2 //移位指令
1e      Lui $5, 0x1111          //R5=0x11110000 //无冲突
1f      Addi $6,$5,0x8000               //R6=0x11108000 符号扩展//2forwarding可以解决
20      J begin                 //j
lable3:
21      Slt $3,$1,$2            //R3=0
22      Slti $4,$3,0x8000       //R4=0  //2forwarding可以解决
23      Andi $5,$6,0x11         //R5=1
24      Jr $31                  //回到1d
```

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