计算机组成原理第六次实验报告（2）

13349051 计科一班 劳嘉辉

实验目的

在实验六（1）的基础上进一步优化 pipeline processor。实现解决 data hazard 的 data forward 机制以及解决 load-use 的方法。

实验难点

1、 实现 data forward 的机制

2、 实现检测 data hazard 的机制

3、 实现解决 load-use 的机制

实验测试

在测试例子文件中提供了两个 ram.vhdl 文件，每个 vhdl 文件都已经包含了 instruction memory 和 data memory 的实现代码。然后 example.txt 包含了汇编指令。在 machine.txt 包含了机器码。使用时，只要在创建的项目里面添加已有的 ram.vhdl 文件即可。

实验过程

1、 实现 data hazard 检测机制

第一步，有一个保存目标寄存器地址的元件：dest_reg

声明：

```
--dest register--------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity dest_reg is
  port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector(2 downto 0);
destin : in std_logic_vector (2 downto 0);
enable : in std_logic;
destoutfir: out std_logic_vector (2 downto 0);
destoutsec: out std_logic_vector (2 downto 0);
destoutthi: out std_logic_vector (2 downto 0)
);
end dest_reg;
```

Opcode: 输入当前指令的 opcode

Destin: 输入当前指令的目标寄存器地址

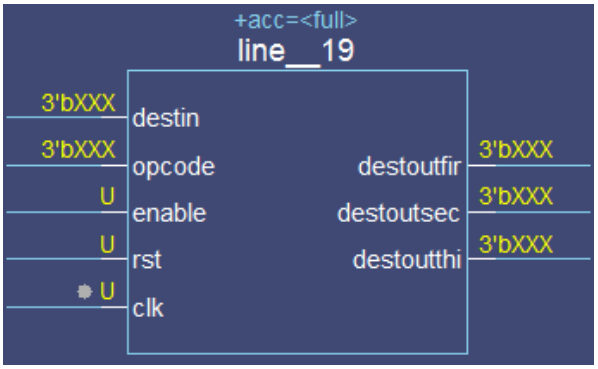
Enable: 使能该寄存器

Destoutfir: 第一个寄存器的地址

Destoutsec: 第二个寄存器的地址

Destoutthi: 第三个寄存器的地址

线路图



实现图

```
architecture bhv of dest_reg is
begin
process(clk)
variable temp1:std_logic_vector (2 downto 0);
variable temp2:std_logic_vector (2 downto 0);
variable temp3:std_logic_vector (2 downto 0);
variable temp4:std_logic_vector (2 downto 0);
begin
  if (rst ='1') then
    destoutfir<=temp1;
    destoutsec<=temp2;
    destoutthi<=temp3;
  else
      if(clk'event and clk ='1' and enable = '0') then
       if(opcode = "011") then
          temp3:=temp2;
          temp2:=temp1;
          temp1:=temp4;
          destoutfir<=temp1;
          destoutsec<=temp2;
          destoutthi<=temp3;

       else
          temp3:=temp2;
          temp2:=temp1;
          temp1:=destin;
          destoutfir<=temp1;
          destoutsec<=temp2;
          destoutthi<=temp3;
        end if;
      elsif(clk'event and clk ='1' and enable = '1')then
      temp3:=temp2;
      temp2:=temp1;
      temp1:=temp4;
      destoutfir<=temp1;
      destoutsec<=temp2;
      destoutthi<=temp3;
      end if;
end if;
end process;
end bhv;
```

说明：

1、 该寄存器用 temp1,temp2,temp3 这三个变量储存寄存器的地址，如果寄存器 enable=0（这是使能）且处于上升沿，那么就输出相应的值.如果 opcode= sw 的话，这里就不会存进 sw 指令的“目标寄存器”，因为 sw 指令的“目标寄

存器”的值并不影响下面寄存器的运算。而对于 load 指令，这里的目标寄存器 regb 由于处于 18-16 的位置被当做 sourcereg，不过这里在 data hazard 原件进行了处理，不会出现多余的 stall。

2、 寄存器的使能是为了处理在 load-use 的时候出现的 stall 的情况，这里给 destoutfir 一个无效的值，这样在 data hazard 判断的时候就不会出错。

3、 寄存器使能的时候使用 temp 给寄存器内的值移位。

第二步，检测 data hazard 的元件 ： Data_hazard

声明

```
--data hazard detect-------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity data_hazatd_detect is
port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector(2 downto 0);
thisopcode:in std_logic_vector(2 downto 0);
sourcerega: in std_logic_vector(2 downto 0);
sourceregb: in std_logic_vector(2 downto 0);
destrega: in std_logic_vector(2 downto 0);
destregb: in std_logic_vector(2 downto 0);
loadhazard: out std_logic;
hazard1: out std_logic;
hazard2: out std_logic;
hazard3: out std_logic;
hazard4: out std_logic
);
end data_hazatd_detect;
```

Opcode：上一个指令的 opcode

Thisopcode：该指令的 opcode

Sourcereg：该指令的两个源寄存器

Destreg：在 dest_Reg 的前两个目标寄存器

Loadhazard：输出是否有 load-use hazard

Hazard：输出四种情况的 hazard

实现：

```
architecture bhv of data_hazatd_detect is
begin
process(clk)
variable  temp1: std_logic:='0';
variable  temp2: std_logic:='0';
begin
  if (rst ='1') then
    hazard1<='0';
    hazard2<='0';
    hazard3<='0';
    hazard4<='0';
    loadhazard<='0';
  else
        if(sourcerega = destrega) then
           hazard1<='1';
           temp1:='1';
         else
           hazard1<='0';
           temp1:='0';
        end if;

        if(sourceregb = destrega) then
          if(thisopcode ="010") then
            hazard2<='0';
            temp2:='0';
          else
            hazard2<='1';
            temp2:='1';
           end if;
          else
            hazard2<='0';
            temp2:='0';
          end if;
        if(sourcerega = destregb) then
            hazard3<='1';
          else
            hazard3<='0';
          end if;
        if(sourceregb = destregb) then
            if(thisopcode ="010") then
            hazard4<='0';
            else
```

```
            hazard4<='1';
           end if;
        else
            hazard4<='0';
          end if;

        if(opcode ="010") then
           if(temp1 = '1' or temp2 ='1') then
              loadhazard<='1';
            else
              loadhazard<='0';
            end if;
        else
           loadhazard<='0';
         end if;
end if;
end process;
end bhv;
```

说明：

1、 分别将当前指令的 sourcerega 和 sourceregb 与已经放在 dest_reg 的 destrega 和 destregb 比较。如果相等的话相应的 hazard 信号就会变成 1，然后该信号会传到 excute 阶段进行处理。

2、 如果 destrega 和 sourcerega 与 sourcereb 任意一个相等，且上一个指令为 load 指令，则判断出是 load-use 的情况，这时候 loadhazard 就会变成 1，然后进行相应的处理。

2、 处理 data hazard

第一步，实现 data_forward

这里在 excute 阶段添加了三个多选器

U_EX_MUX_1:mux4 port map(rst,REGdata,EX_MEM_ALU,IF_EX_REG_vala, irrevalant,EX_MUX_control_1,EX_MUX_OUT_1);
U_EX_MUX_2:mux4port map(rst,IF_EX_REG_valb,REGdata,EX_MEM_ALU,offset_out,EX_MUX_control_2,EX_MUX_OUT_2);
U_EX_MUX_3: mux4 port map(rst,IF_EX_REG_valb,REGdata,EX_MEM_ALU,offset_out,EX_MUX_control_3,EX_MUX_OUT_3);

第一多选器选择的是 val-a 的值。如果 hazard1 为 1 时(destrega =souceregа)，那么就会选择 memory 阶段的 ALU 的值给 ALU。如果 hazard3 为 1 时(destregb=sourcerega),那么就会选择 writeback 阶段的 REGDATA 给 ALU。如果没有 hazard，那么就输出原有的 IF_EX_REG_VALA。

第二个多选器选择的是 val-b 的值。如果为 load 或者 sw 或者 beq 指令，那么必定选择 offse_out 给 ALU。如果 hazard2 为 1 时(destrega =souceregb)，那么就会选择 memory 阶段的 ALU 的值给 ALU。如果 hazard4 为 1 时(destregb=sourceregb),那么就会选择 writeback 阶段的 REGDATA 给 ALU。

如果没有 hazard，那么就输出原有的 IF_EX_REG_VALB。

第三个多选器选择的也是 val-b 的值，这个多选器是为了 sw 这个指令准备的，因为在 sw 的指令的时候，第二个多选器输出的是 offset，因此这里真正的 valb 的值并没有输出到下一个阶段，所以这里的第三个寄存器可以保证 sw 的时候储存到内存的是一个正确的值。

第二步，实现 load-use 的拖延

在上面提及到，如果出现了 load-use 的 hazard，由于在 memory 阶段 pipeline 并不能及时把值传给 ALU，所以这里有一个 stall。Stall 的步骤包括五个：

1） 停住 program control。让 enable 为 0 的时候不输出 PC+1 的值。

2） 停住 memory。让 enable 为 0 的时候继续输出当前指令，而不是输出下一个指令。

3） 停住 IF_ID_REG。让 enable 为 0 的时候继续输出当前指令，而不是输出下一个指令。

4） 把一个无效的 destreg 地址传给 dest_reg 元件。方便下一个指令的 hazard detect。

5） 把一个 noop 的指令传到下一个阶段，拖延了一个 cycle。

实验感想

有了上一次的 pipeline 的设计后，这次的设计就简单很多了。因为这次实验主要在于检测 data hazard 和 在 load-use 的时候实现拖延一个周期。Data hazard 的处理方法并不难，只要添加三个多选器并且根据正确的信号控制输出即可。这里只要改进下 excute_control 即可。但是 load-use 的实现相对难一些，因为停住的一个的话，有相当多的部件也需要正确的被拖延，而且还要在恰当的时间恢复正常，这确实是个难点，不过还是很高兴最后能做出来。

总的来说这几次的实验都让我深入地了解了 VHDL 语言以及 LC2K processor 的机制。虽然在实验过程中遇到很多难点，不过真的学习到了很多东西。