计算机组成原理第六次实验报告（1）
13349051 计科一班 劳嘉辉

实验目的

本次实验是利用 VHDL 语言实现 LC2K 的 pipeline processor。

在第一个报告中，仅实现 pipeline 的五个 stage register 以及 pipeline controller 。

在本次实验中并没有处理 data hazard 以及 detect load instruction 的能力。

实验问题

1、 datapath 的重新设计

2、 pipeline 五个 stage register 的设计

3、 controller 的设计

4、 instruction memory 以及 data memory 的设计

实验调试

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、 新版本的 instruction + data memory 的实现

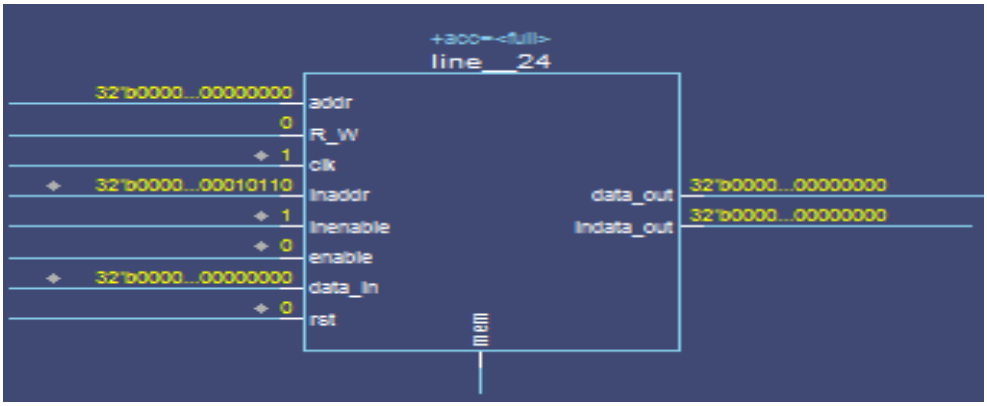
声明部分：

```
--module module-----------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity data_mem is
port(rst :in std_logic;
  clk: in std_logic;
enable : in std_logic;
R_W : in std_logic;
addr : in std_logic_vector (31 downto 0);
data_in : in std_logic_vector (31 downto 0);
data_out : out std_logic_vector (31 downto 0);
inenable : in std_logic;
inaddr : in std_logic_vector (31 downto 0);
indata_out : out std_logic_vector (31 downto 0));
end data_mem;
```

实现部分

```
begin
process(rst,data_in,enable,inenable,inaddr,clk)
variable temp:std_logic_vector (31 downto 0):=conv_std_logic_vector(0,32);
  begin
    if (rst ='1') then
     data_out <=conv_std_logic_vector(0,32);
     indata_out <=conv_std_logic_vector(0,32);
    else
         if(R_W='0' and enable ='1' and clk'event and clk ='0') then
               data_out<=mem(conv_integer(addr));
        elsif (R_W ='1' and enable  ='1' and clk'event and clk ='1') then
               mem(conv_integer(addr))<=data_in;
         end if;

         if(inenable='1' and clk'event and clk ='1') then
               indata_out<=mem(conv_integer(inaddr));
         end if;
   end if;
end process;
end bhv;
```



由于是 pipeline，所以 memory 分为了 data 和 instruction 两部分的内存。

addr: data memory 的地址接口

R_W: data memory 的读写控制接口

Enable: data memory 的使能控制接口

Data_in: data memory 的数据读入接口

Data_out: data memory 的数据输出接口

Inaddr: instruction memory 的地址接口

Inenable: instruction memory 的使能接口

Indata_out: instruction memory 的数据输出接口

用 if-else 语句判断 memory 的读入和输出，对于 data memory 来说，输出数据的时候在下降沿的时候等到控制信号稳定的时候输出较为安全。输入数据的时候在上升沿的时候需要立刻写进，以免发生 data hazard。

2、 Pipeline stage register

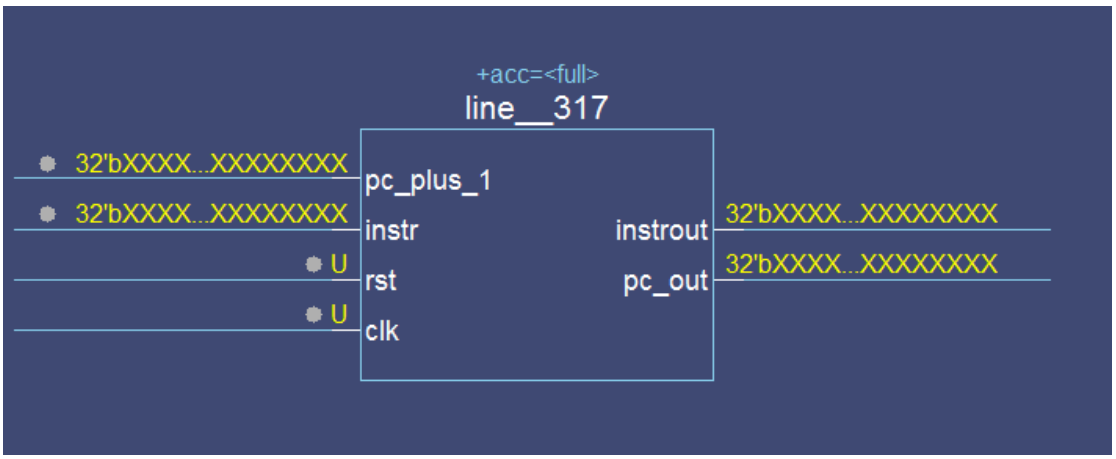
Fetch_decode Stage register : IF/ID 寄存器

声明部分

```
--IF/ID register------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity IF_ID_REG is
  port(
    clk: in std_logic;
    rst: in std_logic;
    instr: in std_logic_vector( 31 downto 0);
    pc_plus_1: in std_logic_vector( 31 downto 0);
    instrout: out std_logic_vector( 31 downto 0);
    pc_out : out std_logic_vector( 31 downto 0));
  end IF_ID_REG;
```

实现部分

```
architecture bhv of IF_ID_REG is
begin
process(clk,rst,instr,pc_plus_1)
  variable temp1 : std_logic_vector (31 downto 0):=conv_std_logic_vector(0,32);
  variable temp2 : std_logic_vector (31 downto 0):=conv_std_logic_vector(0,32);
    begin
      if(rst ='1') then
       instrout<=temp1;
       pc_out<=temp2;
      else
        if(clk'event and clk='1') then
        instrout<=temp1;
        pc_out<=temp2;
        else
        temp1:=instr;
        temp2:=pc_plus_1;
        end if;
    end if;
    end process;
end bhv;
```



说明：

Pc_plus_1： 输入 pc+1 的结果

Instr：输入 instruction

Instrout:在下一个上升沿输出 instruction

Pc_out: 在下一个上升沿输出 pc+1 的结果

这里的做法很简单，其实就是上升沿还没到达的时候，先将结果存到 temp 中，然后在下一个上升沿来到的时候将结果输出。

Decode_excute register : ID/EX 寄存器

声明

```
--ID_EX REG-------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity ID_EX_REG is
  port(
    clk: in std_logic;
    rst: in std_logic;
    firstin: in std_logic_vector( 31 downto 0);
    secondin: in std_logic_vector( 31 downto 0);
    thirdin: in std_logic_vector( 31 downto 0);
    offsetin: in std_logic_vector( 15 downto 0);
    destin: in std_logic_vector ( 2 downto 0);
    opcodein: in std_logic_vector ( 2 downto 0);

    firstout: out std_logic_vector( 31 downto 0);
    secondout: out std_logic_vector( 31 downto 0);
    thirdout: out std_logic_vector( 31 downto 0);
    offsetout: out std_logic_vector( 15 downto 0);
    destout: out std_logic_vector( 2 downto 0);
    opcodeout : out std_logic_vector( 2 downto 0));
  end ID_EX_REG;
```

实现跟上面的 IF/ID 寄存器类似，这里不再重复。

Excute_memory register : EX/MEM 寄存器

声明

```
--EX_MEM_REG-------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity EX_MEM_REG is
  port(
    rst: in std_logic;
    clk: in std_logic;
    targetin: in std_logic_vector( 31 downto 0);
    equalin: in std_logic;
    aluresultin: in std_logic_vector( 31 downto 0);
    valbin: in std_logic_vector( 31 downto 0);
    destin: in std_logic_vector ( 2 downto 0);
    opcodein: in std_logic_vector ( 2 downto 0);

    targetout: out std_logic_vector( 31 downto 0);
    equalout: out std_logic;
    aluresultout: out std_logic_vector( 31 downto 0);
    valbout: out std_logic_vector( 31 downto 0);
    destout: out std_logic_vector( 2 downto 0);
    opcodeout : out std_logic_vector( 2 downto 0));
  end EX_MEM_REG;
```

Memory_write_back register : MEM/WB 寄存器

声明

```
--MEM_WB_REG-------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity MEM_WB_REG is
  port(
    clk: in std_logic;
    rst: in std_logic;
    firstin: in std_logic_vector( 31 downto 0);
    secondin: in std_logic_vector( 31 downto 0);
    destin: in std_logic_vector ( 2 downto 0);
    opcodein: in std_logic_vector ( 2 downto 0);

    firstout: out std_logic_vector( 31 downto 0);
    secondout: out std_logic_vector( 31 downto 0);
    destout: out std_logic_vector( 2 downto 0);
    opcodeout : out std_logic_vector( 2 downto 0));
  end MEM_WB_REG;
```

3、 pipeline stage controller

fetch_controller

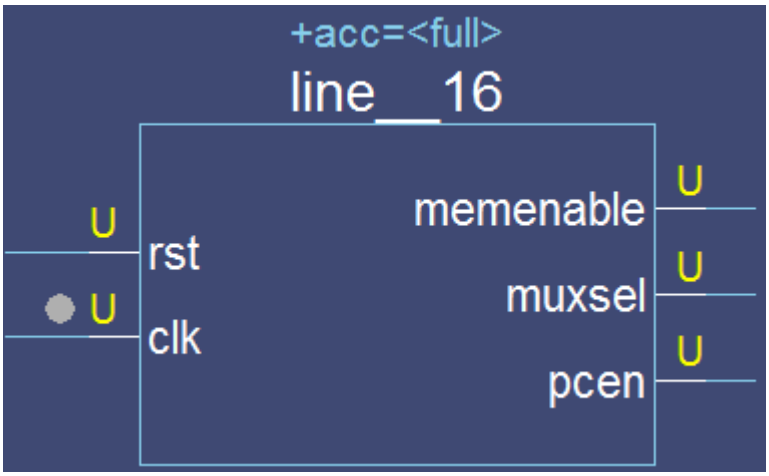
声明

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity fetch_control is
port(
rst: in std_logic;
clk: in std_logic;
pcen : out std_logic;
memenable: out std_logic;
muxsel : out std_logic
);
```

实现

```
architecture bhv of fetch_control is
begin
   process(clk)
     begin
       if(rst = '1') then
          pcen<='1';
          memenable<='1';
          muxsel<='1';
        else
       pcen<='1';
       memenable<='1';
       muxsel<='1';
    end if;
end process;
end bhv;
```

线路图



decode_controller

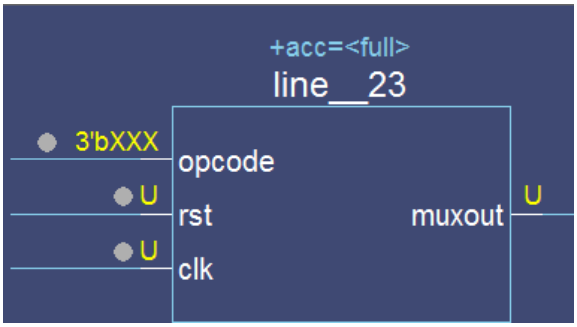
声明

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity decode_control is
port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector( 2 downto 0);
muxout: out std_logic
);
end decode control;
```

实现图

```
architecture bhv of decode_control is
constant ADD : std_logic_vector(2 downto 0) := "000";
constant NAD : std_logic_vector(2 downto 0) := "001";
constant LW : std_logic_vector(2 downto 0) := "010";
constant SW : std_logic_vector(2 downto 0) := "011";
constant BEQ : std_logic_vector(2 downto 0) := "100";
constant JALR : std_logic_vector(2 downto 0) := "101";
constant HALT : std_logic_vector(2 downto 0) := "110";
constant NOOP : std_logic_vector(2 downto 0) := "111";
begin
   process(clk,rst,opcode)
     begin
       if(rst ='1') then
          muxout<='0';
        else
          if(opcode = ADD or opcode =NAD) then
             muxout<='0';
          elsif(opcode = HALT) then
             muxout<='1';
         else
             muxout<='1';
       end if;
      end if;
end process;
```

线路图



说明：

Opcode：输入来自 IF/ID register 的 opcode

Muxout: 输出对 dest_mux 的控制信号

如果是 add 和 nand 的话，输出为“0”，选择“2-0”

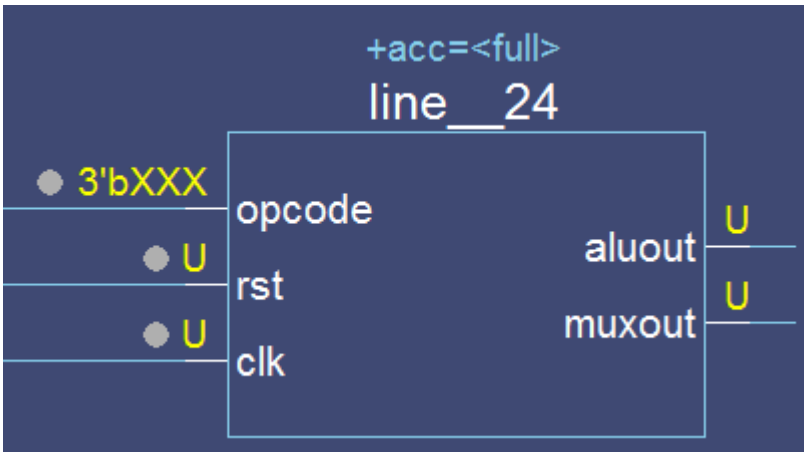
如果是其他的话，输入为“1”，选择“18-16”或者无关

excute_controller

声明

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity excute_control is
port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector( 2 downto 0);
muxout: out std_logic;
aluout: out std_logic
);
```

线路



Opcode: 输入的是 opcode 信号

Aluout: 输出控制 ALU 操作的信号

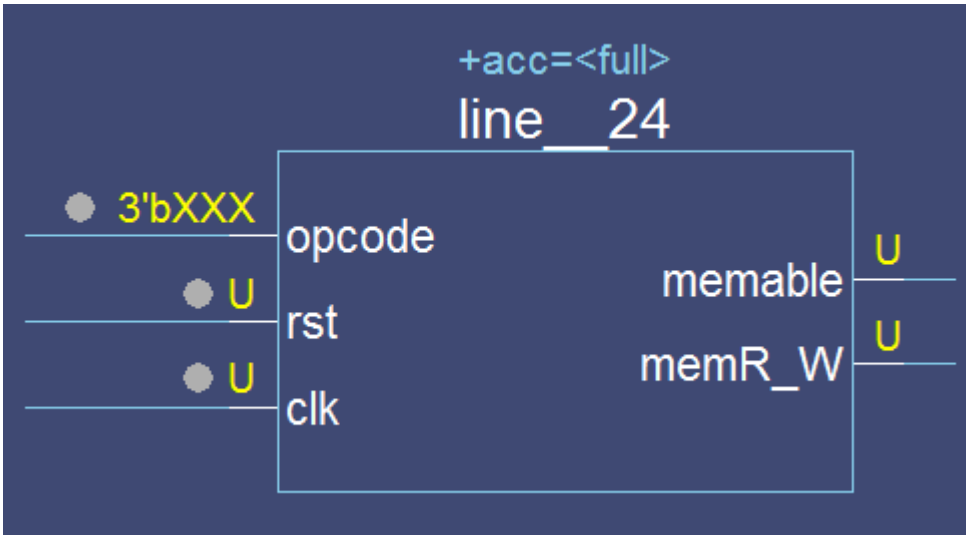
Muxout: 输出控制多选器的信号

memory_controller

声明

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity memory_control is
port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector( 2 downto 0);
memable: out std_logic;
memR_W : out std_logic
);
end memory_control;
```

线路图



Memable: 输出的是控制 data memory 使能的信号

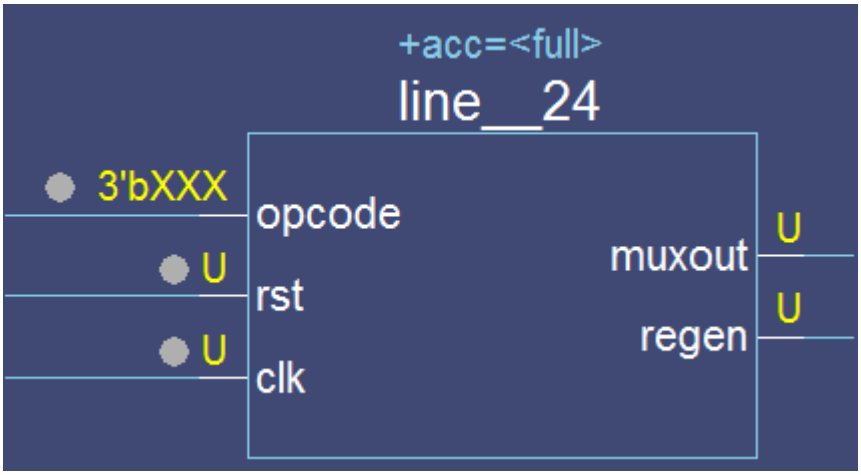
memR_W: 输出的是控制 data memory 读写的信号

Write_back_controller

声明

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity write_back_control is
port(
rst: in std_logic;
clk: in std_logic;
opcode: in std_logic_vector( 2 downto 0);
muxout: out std_logic;
regen : out std_logic
);
end write back control;
```

线路图



说明：

Muxout:输出控制返回写入的多选器信号

Regen:输出控制寄存器堆的信号

4、 datapath 连接

```
U_PC : reg port map (rst,clk,PCen,IF_MUX_OUT,PCout);
U_IF_ADDER : adder port map(PCout,conv_std_logic_vector(1,32),pc_1_out);
U_IF_MUX : mux2 port map(rst,target,pc_1_out,IF_mux,IF_MUX_OUT);
U_IF_ID_REG: IF_ID_REG port map(clk,rst,Inst_mem_out,pc_1_out,IF_ID_REG_inst_out,IF_ID_REG_pc_out);
U_fetch_control : fetch_control port map(rst,clk,PCen,Inst_mem_en,IF_mux);
U_REG_bank : register_file port map (rst,clk,REGen, REGW, IF_ID_REG_inst_out(21 downto 19), IF_ID_REG_inst_out(18 downto 16),
REGdata, REGout1, REGout2,r0,r1,r2,r3,r4,r5,r6,r7);
U_ID_MUX: muxr port map(rst,IF_ID_REG_inst_out(2 downto 0), IF_ID_REG_inst_out(18 downto 16),ID_MUX_control ,ID_MUX_OUT);
U_ID_EX_REG :ID_EX_REG port map(clk,rst,IF_ID_REG_pc_out,REGout1,REGout2,IF_ID_REG_inst_out(15 downto 0),
  ID_MUX_OUT,IF_ID_REG_inst_out(24 downto 22),IF_EX_REG_pc_out,IF_EX_REG_vala,IF_EX_REG_valb,IF_EX_REG_offset
,IF_EX_REG_dest,IF_EX_REG_opcode);
U_decode_control: decode_control port map(rst,clk,IF_ID_REG_inst_out(24 downto 22),ID_MUX_control);
U_sext: sext port map(rst,IF_EX_REG_offset,offset_out);
U_EX_ADDER: adder port map(IF_EX_REG_pc_out,offset_out,EX_ADDER_OUT);
U_EX_MUX: mux2 port map(rst,IF_EX_REG_valb,offset_out,EX_MUX_control,EX_MUX_OUT);
U_EX_ALU: LC2_ALU port map(rst,IF_EX_REG_vala,EX_MUX_OUT,ALUsel,ALUout,ALUeql);
U_EX_MEM_REG: EX_MEM_REG port map(rst,clk,EX_ADDER_OUT,ALUeql,ALUout,IF_EX_REG_valb,IF_EX_REG_dest,IF_EX_REG_opcode,
  target,EX_MEM_eq,EX_MEM_ALU,EX_MEM_valb ,EX_MEM_dest,EX_MEM_opcode);
U_excute_control : excute_control port map(rst,clk,IF_EX_REG_opcode,EX_MUX_control,ALUsel);□
U_data_mem : data_mem port map(rst,clk,data_mem_enable,data_mem_R_W,EX_MEM_ALU,EX_MEM_valb ,data_mem_out,Inst_mem_en,PCout,Inst_mem_out);
U_MEM_WB_REG: MEM_WB_REG port map(clk,rst,EX_MEM_ALU,data_mem_out, EX_MEM_dest,EX_MEM_opcode,MEM_WB_ALU,MEM_WB_mdata,REGW,MEM_WB_opcode);
U_memory_control : memory_control port map(rst,clk,EX_MEM_opcode,data_mem_enable,data_mem_R_W);
U_WB_MUX: mux2 port map(rst,MEM_WB_ALU,MEM_WB_mdata,WB_MUX_sel,REGdata);
U_write_back_control : write_back_control port map(rst,clk,MEM_WB_opcode,WB_MUX_sel,REGen);
```

U_PC : reg port map (rst,clk,PCen,IF_MUX_OUT,PCout);

U_IF_ADDER : adder port map(PCout,conv_std_logic_vector(1,32),pc_1_out);

U_IF_MUX : mux2 port map(rst,target,pc_1_out,IF_mux,IF_MUX_OUT);

U_IF_ID_REG: IF_ID_REG port map(clk,rst,Inst_mem_out,pc_1_out,IF_ID_REG_inst_out,IF_ID_REG_pc_out);

U_fetch_control : fetch_control port map(rst,clk,PCen,Inst_mem_en,IF_mux);

U_REG_bank : register_file port map (rst,clk,REGen, REGW, IF_ID_REG_inst_out(21 downto 19),

IF_ID_REG_inst_out(18 downto 16),REGdata, REGout1, REGout2,r0,r1,r2,r3,r4,r5,r6,r7);

U_ID_MUX: muxr port map(rst,IF_ID_REG_inst_out(2 downto 0), IF_ID_REG_inst_out(18 downto 16),ID_MUX_control ,ID_MUX_OUT);

U_ID_EX_REG :ID_EX_REG port map(clk,rst,IF_ID_REG_pc_out,REGout1,REGout2,IF_ID_REG_inst_out(15 downto 0),ID_MUX_OUT,IF_ID_REG_inst_out(24downto22),IF_EX_REG_pc_out,IF_EX_REG_vala,IF_EX_REG_valb,IF_EX_REG_offset,IF_EX_REG_dest,IF_EX_REG_opcode);

U_decode_control: decode_control port map(rst,clk,IF_ID_REG_inst_out(24 downto 22),ID_MUX_control);

U_sext: sext port map(rst,IF_EX_REG_offset,offset_out);

U_EX_ADDER: adder port map(IF_EX_REG_pc_out,offset_out,EX_ADDER_OUT);

U_EX_MUX: mux2 port map(rst,IF_EX_REG_valb,offset_out,EX_MUX_control,EX_MUX_OUT);

U_EX_ALU: LC2_ALU port map(rst,IF_EX_REG_vala,EX_MUX_OUT,ALUsel,ALUout,ALUeql);

U_EX_MEM_REG:EX_MEM_REGportmap(rst,clk,EX_ADDER_OUT,ALUeql,ALUout,IF_EX_REG_valb,IF_EX_REG_dest,IF_EX_REG_opcode,target,EX_MEM_eq,EX_MEM_ALU,EX_MEM_valb ,EX_MEM_dest,EX_MEM_opcode);

U_excute_control : excute_control port map(rst,clk,IF_EX_REG_opcode,EX_MUX_control,ALUsel);

U_data_mem :data_memportmap(rst,clk,data_mem_enable,data_mem_R_W,EX_MEM_ALU,EX_MEM_valb ,data_mem_out,Inst_mem_en,PCout,Inst_mem_out);

U_MEM_WB_REG: MEM_WB_REG port map(clk,rst,EX_MEM_ALU,data_mem_out, EX_MEM_dest,EX_MEM_opcode,MEM_WB_ALU,MEM_WB_mdata,REGW,MEM_WB_opcode);

U_memory_control : memory_control port map(rst,clk,EX_MEM_opcode,data_mem_enable,data_mem_R_W);

U_WB_MUX: mux2 port map(rst,MEM_WB_ALU,MEM_WB_mdata,WB_MUX_sel,REGdata);

U_write_back_control : write_back_control port map(rst,clk,MEM_WB_opcode,WB_MUX_sel,REGen);

实验感想

这次实验比想象中的更有难度。本次实验难点不在实现四个寄存器，因为这四个寄存器的实现方式是类似的。

难点在于内存和寄存器堆的重新实现。为了配合 pipeline 的实现，memory 和 register file 的读取和输出都要在更加具体的情况下才能够正确读取和输出。在这里 debug 了很久，不过最后找到了方法也是挺好的。

还有就是敏感列表的使用，敏感列表的参数过多，或者应该说加入了不应该有的参数，就会导致 memory 和 register file 读取和输出的错误。