RISC-V 基本指令集 CPU 设计与实现

班级：智能 1602 学号：201608010627 姓名：任小禹

一、实验目标

完成一个执行 RISC-V 的基本整数指令集 RV32I 的 CPU 设计

二、实验要求

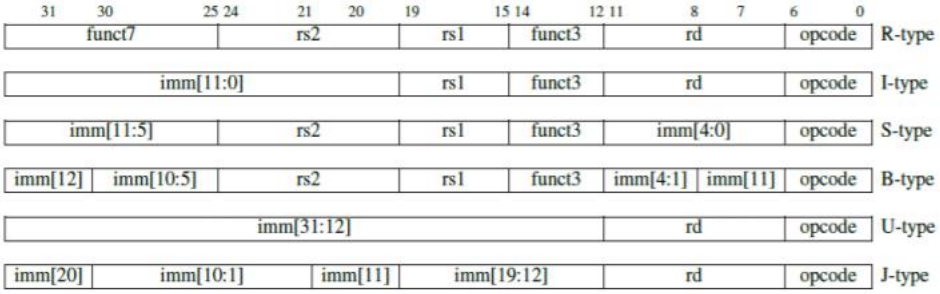
1、采用 VHDL 或 Verilog 语言编写

2、采用时序逻辑设计电路

三、实验内容

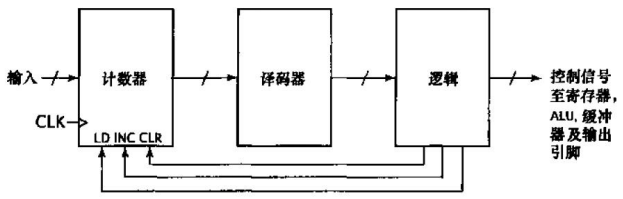
1、指令集

基本指令集共有 47 条指令。除去系统指令我们这里实现 37 条。一共 6 类指令，各类型指令格式：



2、设计思路

- 我们设计 5 步设计单周期 cpu，指令的执行过程是取指令，译码，执行，访存，写回。
- CPU 的控制单元采用简单硬布线控制：

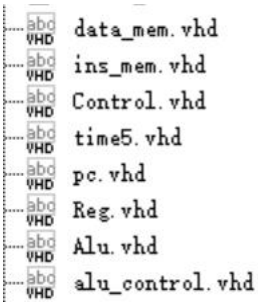


- 设计步骤：

ALU 的设计->存储器的设计->整合 ALU、存储器进行数据通路的设计->时序逻辑单元的设计->控制单元的设计->顶层文件设计，整合整个 CPU

3、程序清单

- 顶层设计文件：rxycpu.vhd
- 其它设计文件：



前两个是存储器模块，包括指令存储器和数据存储器。

控制单元模块：

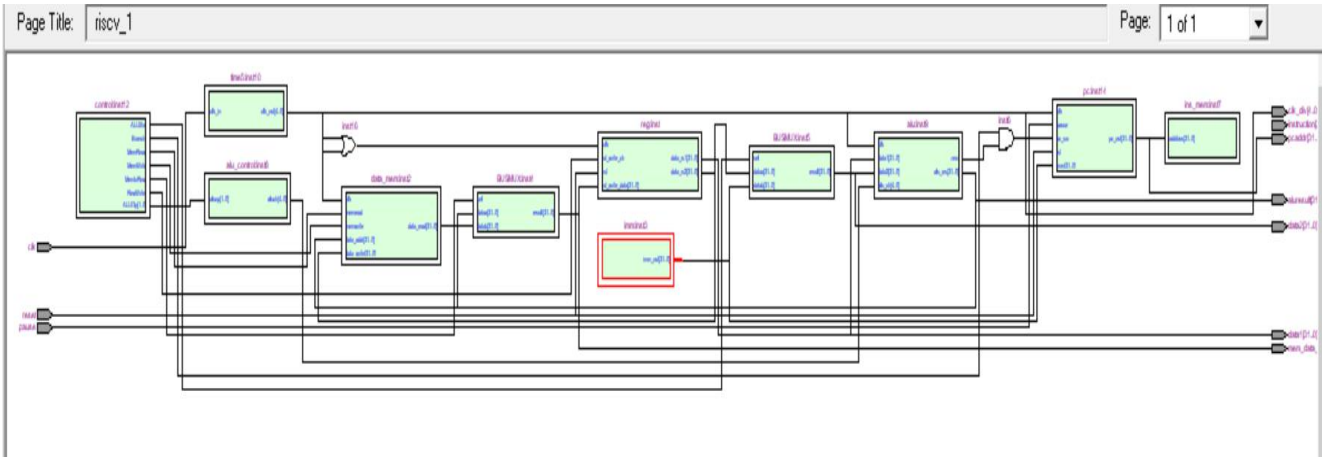
Control.vhd 和 alu_control.vhd 是控制信号模块。

time5.vhd 是时序逻辑单元模块。pc.vhd 是产生下址的单元模块。

其余部分对应相应的功能。

4、程序框架

使用 Quartus II 自带的 RTL Viewer 查看顶层关系图:



顶层设计文件主要是元件例化语句，将各个模块连接起来，定义元件与元件、元件与其它信号、元件与外部信号端口的连接关系。例如：

```
COMPONENT alu
    PORT(clk : IN STD_LOGIC;
         alu_ctr : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
         data1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         data2 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         zero : OUT STD_LOGIC;
         alu_res : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END COMPONENT;
```

四、实验测试

1、测试内容

设置合理的测试指令序列，进行波形仿真。

设计的原则：

保证每类指令都可以测试到，并且可以对比符号扩展和零扩展。机器为小端，测试时将指令码的高字节保存在内存的高地址中。

2、测试平台

我的 CPU 在如下机器上进行了测试

部件	配置	备注
CPU	core i7-8500U	
内存	DDR4 8GB	
操作系统	Windows 10	家庭中文版
综合软件	Quartus II 9.0sp1 Web Edition	
仿真软件	Quartus II 9.0sp1 Web Edition 自带仿真器	
波形查看	Quartus II 9.0sp1 Web Edition Simulate Report	

3、测试输入

指令存储器存储以下指令：

```
0 => X"83",
1 => X"00",
2 => X"00",
3 => X"00",

4 => X"03",
5 => X"41",
6 => X"10",
7 => X"00", --

8 => X"b3",
9 => X"81",
10=> X"20",
11=> X"00",

12=> X"37",
13=> X"F0",
14=> X"FF",
15=> X"FF",
```

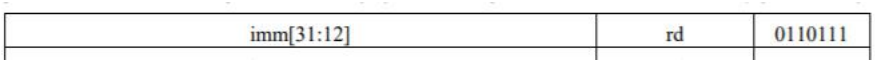
数据存储器存储以下数据：

```
0 => X"F3",
1 => X"F2",
2 => X"01",
3 => X"80",
```

4、测试记录

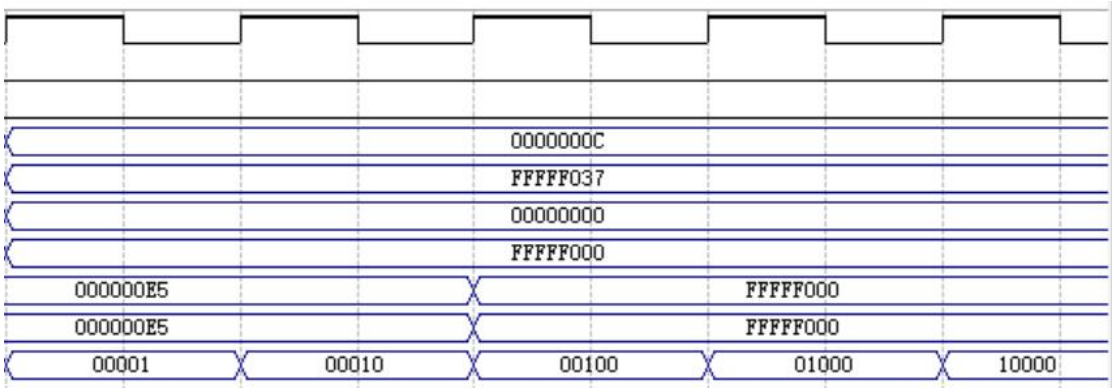
1）LUI 指令

将立即数作为高 20 位，低 20 位用 0 填充，结果放进 RD 寄存器



立即数 0xfffff，rd 寄存器为 0x0，指令码为 0xfffff037。

测试结果：



2）LB 指令

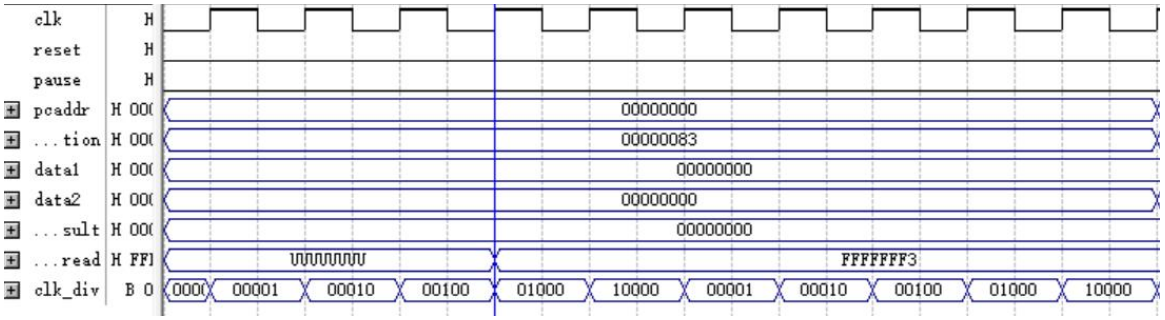
LB 将指令高 12 位作为立即数有符号扩展与 R1 的值相加，作为地址，读取该地

址的一个字节，有符号扩展为 32 位，并存入 RD 寄存器

imm[11:0]	rs1	000	rd	0000011	I lb

000000000000 00000 000 00001 0000011

指令码为 0x00000083,立即数为 0x0，rs1 寄存器为 0 号，rd 寄存器为 1 号



读取地址 0 的一个字节 0xf3，有符号拓展为 0xfffffff3。我们也可以看到 0xfffffff3 出现，刚好在指令的访存阶段，第四阶段（01000B)。

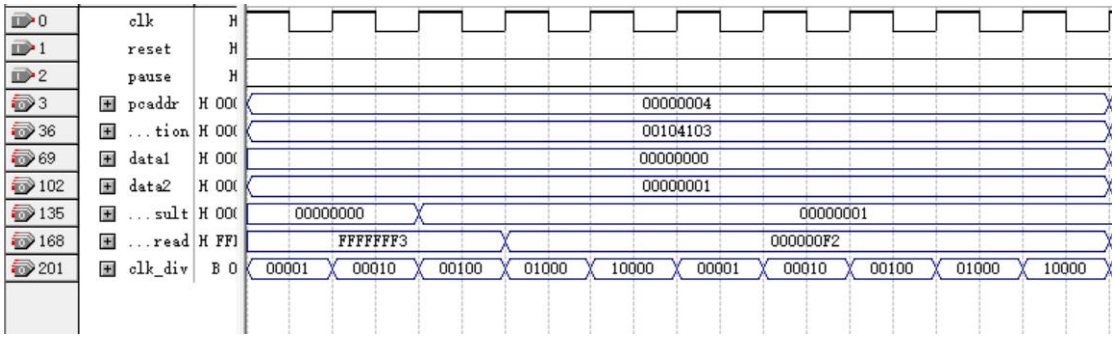
3）LBU 指令

LBU 将指令高 12 位作为立即数有符号扩展与 R1 的值相加，作为地址，读取该地址的一个字节，零扩展为 32 位，并存入 RD 寄存器

imm[11:0]	rs1	100	rd	0000011	I lbu

000000000001 00000 100 00010 0000011

指令码为 0x00104103,立即数为 0x1，rs1 寄存器为 2 号，rd 寄存器为 1 号



读取地址 0 的一个字节 0xf2，零拓展拓展为 0x000000f2，**对比 2）我们可以看出零扩展和符号扩展的区别**。

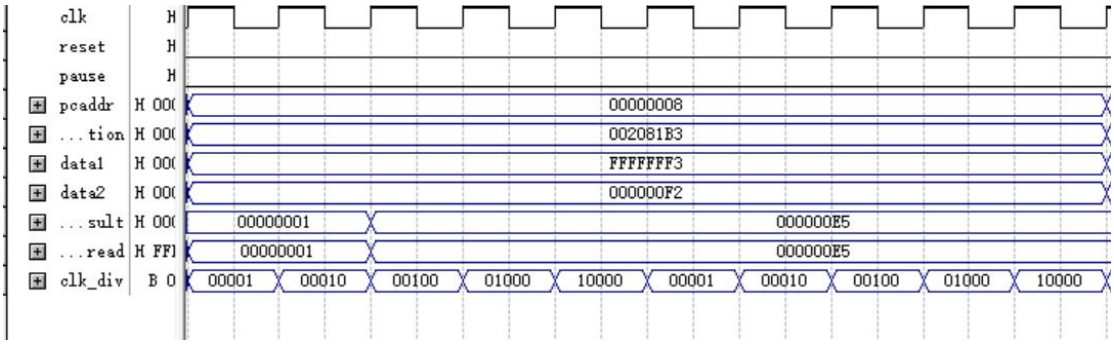
4）ADD 指令

ADD 加操作，RD=R1+R2

0000000	rs2	rs1	000	rd	0110011	R add

0000000 00010 00001 000 00011 0110011

指令码为 0x002081B33,rs2 寄存器为 2 号,rs1 寄存器为 1 号，rd 寄存器为 3 号



我们可以看得最后结果是 rs1 寄存器+rs2 寄存器的值，这时结果溢出，丢弃最高位。

五、分析与结论

从测试记录来看，CPU 实现了对测试程序指令的取指令，译码，执行，访存，写回，得到的运算结果正确。

根据分析结果，可以认为所设计的 CPU 实现了所要求的功能，完成了实验目标。

六、遇到的问题

1、注意实体名不能起 time，因为 time 是特殊字段，在 Quartus II 编译会出错。

```
entity time is
port(
    clk_in:in std_logic;
    clk_out:out std_logic_vector(4 downto 0)
    );
```

2、元件例化语句一定要一一对应，注意**连接关系和中间变量一定要和模块的相应变量的一致**，不然会不停报错。

```
b2v_inst : reg
PORT MAP(clk => SYNTHESIZED_WIRE_0,
         rst => rst,
         rd_write_ctr => rwc,
         rd => ins(11 DOWNTO 7),
         rd_write_data => rwd,
         rs1 => ins(19 DOWNTO 15),
         rs2 => ins(24 DOWNTO 20),
         data_rs1 => SYNTHESIZED_WIRE_10,
         data_rs2 => SYNTHESIZED_WIRE_13);
```

```
entity reg is
port(
    clk:in std_logic;
    rst:in std_logic;
    rs1:in std_logic_vector(4 downto 0);  --rs1
    rs2:in std_logic_vector(4 downto 0);  --rs2

    rd_write_ctr:in std_logic;
    rd:in std_logic_vector(4 downto 0);  --rd
    rd_write_data:in std_logic_vector(31 downto 0)

    data_rs1:out std_logic_vector(31 downto 0);  -
    data_rs2:out std_logic_vector(31 downto 0)   -
    );
end reg;
```

七、收获与体会

编写了很多次 CPU，大体的设计思路已经掌握，从 ALU 入手到最后顶层设计，自己也总结了很多经验，数据通路完成 CPU 就完成一半了，最难的总是时序逻辑和控制单元的部分，最需要细心的永远是最后顶层设计的环节，一不小心就会出错。这次 RISC-V 基本指令集 CPU 的设计，也算是圆满地完成了。在设计的过程中也发现了 RISC-V 基本指令集的很多好处，比如它是定长的，都是 32 位，格式简单，而且 6 类指令各自格式统一，这就给我们设计的带来了很多方便和快捷。CPU 的设计，让我也体会到了符号扩展和零扩展、以及大端小端的差别，使我硬件设计更加熟练，更加熟悉 VHDL 语言，收获很大。