HUNAN UNIVERSITY

RISC-V 基本指令集 CPU 设计与实现

班级：智能 1602

学号：201608010723

姓名：施园

一、实验目的

完成一个模拟 RISC-V 的基本整数指令集 RV32 的 CPU(单周期)设计。

二、实验要求

1、硬件设计采用 VHDL 或 Verilog 语言，软件设计采用 C/C++或 SystemC 语言，其它语言例如 Chisel、MyHDL 等也可选。

2、实验报告采用 markdown 语言，或者直接上传 PDF 文档

3、实验最终提交所有代码和文档。

三、实验过程

这份工作在暑假的时候没有做过，所以本次实验相对简单一点。本实验实现 5 条指令，分别为 LUI，BGE，LBU，SLTIU，SRAI。其中，LUI 存放立即数到 rd 的高 20 位，低 12 位置 0。BGE 在满足 src1>=src2 条件时跳转，跳转范围为 pc(+/-)4KB。LBU 从存储器加载一个 8 位值，然后在存储到 reg[rd]之前将零扩展到 32 位。SLTIU 在 src1 小于立即数(都是无符号整数)的情况下将 reg[rd]置 1,否则置 0。SRAI 将 src1 里面的数据算数右移，并存入 reg[rd]内，入参定义如下，包括了输入输出、时钟、重置等信号。

设计框架如下：

```
entity rv32i_cpu_singlecycle is
port(
clk: in std_logic;
reset: in std_logic;
inst_addr: out std_logic_vector(31 downto 0);
inst: in std_logic_vector(31 downto 0);
```

```
data_addr: out std_logic_vector(31 downto 0);
data: inout std_logic_vector(31 downto 0);
data_read: out std_logic;
data_write: out std_logic
);
end entity rv32i_cpu_singlecycle;
```

结构部分，声明了计算是需要使用的变量。ir 表示当前执行的指令，pc 表当前的指令的地址；7 位的 opcode，3 位的 funct3，7 位的 funct7，这三个变量读取 ir 的指令，取到对应的值。寄存器 rd,rs1,rs2 存储 ir 中读取到的对应操作值地址，src1,src2 将 rs1,rs2 中的地址对于的 reg 中的值转为 32 位保存。

```
signal ir: std_logic_vector(31 downto 0);
signal pc: std_logic_vector(31 downto 0);
signal next_pc: std_logic_vector(31 downto 0);
-- Fields in instruction
signal opcode: std_logic_vector(6 downto 0);
signal rd: std_logic_vector(4 downto 0);
signal funct3: std_logic_vector(2 downto 0);
signal rs1: std_logic_vector(4 downto 0);
signal rs2: std_logic_vector(4 downto 0);
signal funct7: std_logic_vector(6 downto 0);
signal shamt: std_logic_vector(4 downto 0);
signal Imm31_12UtypeZeroFilled: std_logic_vector(31 downto 0);
signal Imm12_1BtypeSignExtended: std_logic_vector(31 downto 0);
signal Imm11_0ItypeSignExtended: std_logic_vector(31 downto 0);
signal src1: std_logic_vector(31 downto 0);
signal src2: std_logic_vector(31 downto 0);
signal addresult: std_logic_vector(31 downto 0);
signal subresult: std_logic_vector(31 downto 0);
type regfile is array(natural range<>) of std_logic_vector(31 downto 0);
signal regs: regfile(31 downto 0);
signal reg_write: std_logic;
signal reg_write_id: std_logic_vector(4 downto 0);
signal reg_write_data: std_logic_vector(31 downto 0);
signal LUIresult: std_logic_vector(31 downto 0);
signal AUIPCresult: std_logic_vector(31 downto 0);
```

```
signal BGEresult: std_logic_vector(31 downto 0);
signal LBUresult: std_logic_vector(31 downto 0);
signal SLTIUresult: std_logic_vector(31 downto 0);
signal SRAIresult: std_logic_vector(31 downto 0);
```

reg_write 为写操作的标记，当为'1'时表示需要将 reg_write_data 的值写入下标为 reg_write_id 的寄存器中。

```
signal reg_write: std_logic;
signal reg_write_id: std_logic_vector(4 downto 0);
signal reg_write_data: std_logic_vector(31 downto 0);
```

获取对于的值：

```
inst_addr <= pc;
ir <= inst;
opcode <= ir(6 downto 0);
rd <= ir(11 downto 7);
funct3 <= ir(14 downto 12);
rs1 <= ir(19 downto 15);
rs2 <= ir(24 downto 20);
funct7 <= ir(31 downto 25);
shamt <= rs2;
Imm31_12UtypeZeroFilled <= ir(31 downto 12) & "000000000000";
Imm12_1BtypeSignExtended <= "1111111111111111111" & ir(31) & ir(7) & ir(30
downto 25) & ir(11 downto 8) when ir(31)='1' else
"0000000000000000000" & ir(31) & ir(7) & ir(30 downto 25) & ir(11 downto 8);
Imm11_0ItypeSignExtended <= "11111111111111111111" & ir(31 downto 20) when
ir(31)='1' else "00000000000000000000" & ir(31 downto 20);
src1 <= regs(TO_INTEGER(UNSIGNED(rs1)));
src2 <= regs(TO_INTEGER(UNSIGNED(rs2)));
reg_write_id <= rd;
addresult <= STD_LOGIC_VECTOR(SIGNED(src1) + SIGNED(src2));
subresult <= STD_LOGIC_VECTOR(SIGNED(src1) - SIGNED(src2));
LUIresult <= Imm31_12UtypeZeroFilled;
AUIPCresult<=STD_LOGIC_VECTOR(SIGNED(pc)+
SIGNED(Imm31_12UtypeZeroFilled));
SRAIresult<=to_stdlogicvector(to_bitvector(src1)SRA
to_integer(unsigned(shamt)) ) ;
SLTIUresult<="00000000000000000000000000000001"when
TO_INTEGER(UNSIGNED(src1))                                    <
TO_INTEGER(UNSIGNED(Imm11_0ItypeSignExtended)) else
"00000000000000000000000000000000";
LBUresult <= "000000000000000000000000" & data_in(7 downto 0);
reg_write_data <= addresult when opcode = "0110011" and funct7 = "0000000" else
subresult when opcode = "0110011" and funct7 = "0100000" else
```

```
LUIresult when opcode = "0110111" else
AUIPCresult when opcode = "0010111" else
LBUresult when opcode = "0000011" and funct3 = "100" else
SRAIresult when opcode = "0010011" and funct3 = "101" and ir(31 downto 25) = "0100000" else
SLTIUresult when opcode = "0010011" and funct3 = "011" else
"00000000000000000000000000000000";
next_pc<=STD_LOGIC_VECTOR(SIGNED(pc) + SIGNED(Imm12_1BtypeSignExtended)) when opcode = "1100011" and funct3 = "101" and SIGNED(src1) >= SIGNED(src2) else
STD_LOGIC_VECTOR(SIGNED(pc) + 4);
```

在执行阶段，根据获取到的值，计算出指令的结果。其中 nextpc 正常情况下+4，在满足 BGE 条件时跳转到对应地址。

```
LUIresult <= Imm31_12UtypeZeroFilled;
AUIPCresult<=TD_LOGIC_VECTOR(SIGNED(pc)+
SIGNED(Imm31_12UtypeZeroFilled));
SRAIresult<=_stdlogicvector(to_bitvector(src1) SRA to_integer(unsigned(shamt)) ) ;
SLTIUresult<=”00000000000000000000000000000001"when
TO_INTEGER(UNSIGNED(src1))<
TO_INTEGER(UNSIGNED(Imm11_0ItypeSignExtended)) else
"00000000000000000000000000000000";
LBUresult <= "000000000000000000000000" & data_in(7 downto 0);
reg_write_data <= addresult when opcode = "0110011" and funct7 = "0000000" else
subresult when opcode = "0110011" and funct7 = "0100000" else
LUIresult when opcode = "0110111" else
AUIPCresult when opcode = "0010111" else
LBUresult when opcode = "0000011" and funct3 = "100" else
SRAIresult when opcode = "0010011" and funct3 = "101" and ir(31 downto 25) = "0100000" else
SLTIUresult when opcode = "0010011" and funct3 = "011" else
next_pc<=STD_LOGIC_VECTOR(SIGNED(pc)+
SIGNED(Imm12_1BtypeSignExtended)) when opcode = "1100011" and funct3 = "101" and SIGNED(src1) >= SIGNED(src2) else
STD_LOGIC_VECTOR(SIGNED(pc) + 4);
```

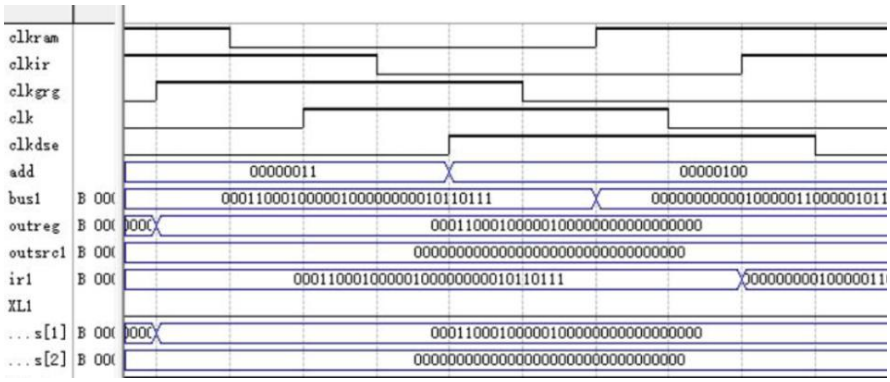
最后写回阶段，当时钟上跳时触发。

```
process(clk)
begin
if(rising_edge(clk)) then
if (reset='1') then
pc <= "00000000000000000000000000000000";
-- Clear register file?
```

```
else
pc <= next_pc;
if (reg_write = '1') then
regs(TO_INTEGER(UNSIGNED(reg_write_id))) <= reg_write_data;
end if; -- reg_write = '1'
end if; -- reset = '1'
end if; -- rising_edge(clk)
end process; -- clk
STD_LOGIC_VECTOR(SIGNED(pc) + 4);
```

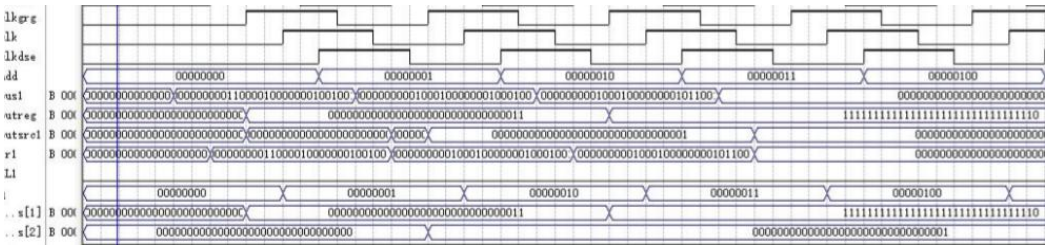
实验测试：

1、对于指令 LUI,当输入指令为 01000000001000010000000010110011 时，测试结果如下:



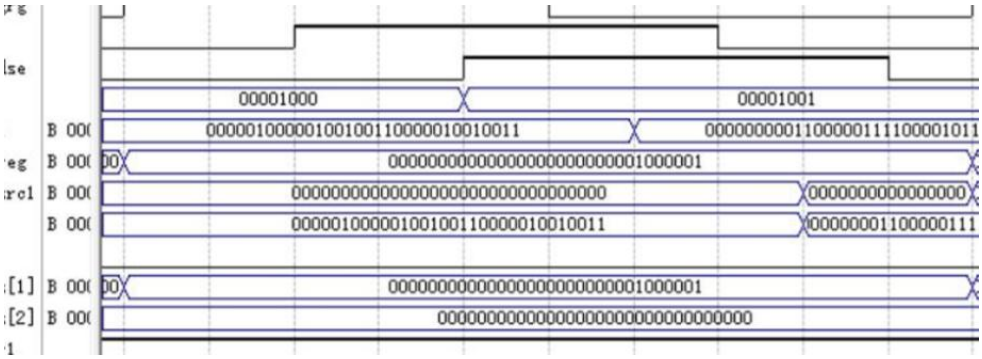
本条指令将 20 位立即数移至寄存器 1 的前 20 位，而寄存器后 12 位置 0。从结果可知，此时立即数已向前移位 12 位，后 12 位置 0，可以看出该指令正确执行。

2、对于指令 BGE,当输入指令为 00011000100000100000000010110111 时，测试结果如下:



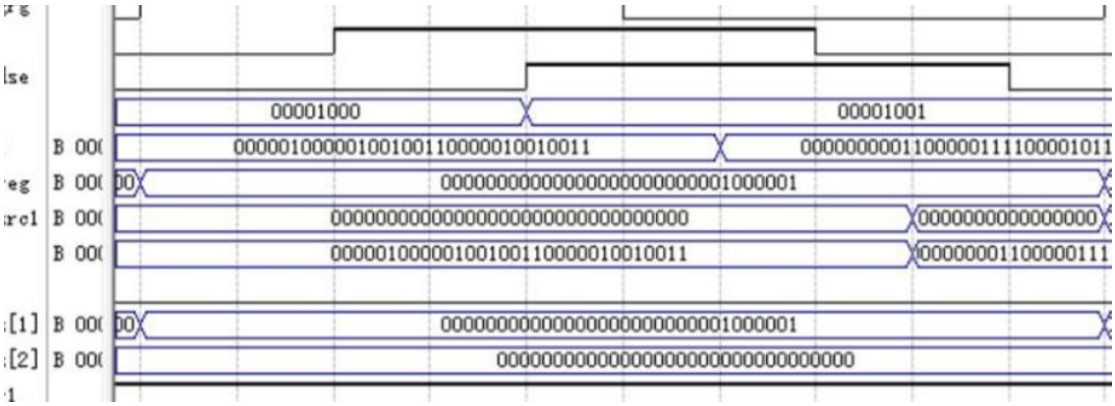
在满足 src1>=src2 条件时发生了跳转。

3、对于指令 BGE,当输入指令为 00011000100000100000000010110111 时，测试结果如下:



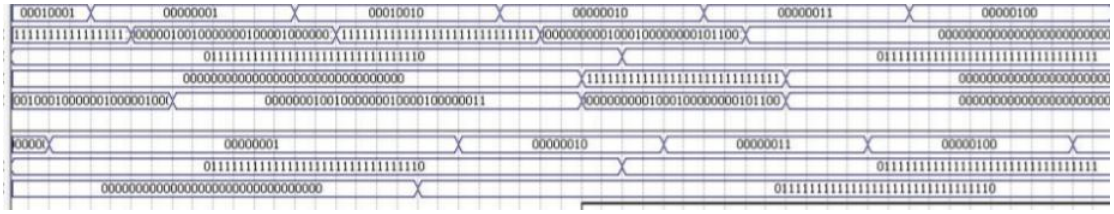
LBU 从存储器加载一个 8 位值，然后在存储到 reg[rd]之前将零扩展到 32 位。

4、对于指令 SRAI,当输入指令为 00000010100101000111000010010011 时，测试结果如下:



SRAI 将 src1 里面的数据算数右移，并存入 reg[rd]内

5、对于指令 SLTIU,当输入指令为 00011000100000100000000010110111 时，测试结果如下:



SLTIU 在 src1 小于立即数(都是无符号整数)的情况下将 reg[rd]置 1,否则置 0。
从测试结果可以看出编写的 cpu 能够完成指令的工作，达到了实验的目的。

四、分析和总结

在将 cpu 整合的过程中，逐步将各个模块合成并且运行起来，虽然 CPU 设计难度较大的，需要花很多的精力和时间在上面，但是这一切都很值得，总的来说收获很多。