# 实验报告

班级：通信1604 姓名：哈拿提·海衣尔别克 学号：201608030429

## 实验内容

实现单周期CPU的设计。

## 实验要求

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。

实验报告采用markdown语言，或者直接上传PDF文档

实验最终提交所有代码和文档

## 模拟环境

部件 配置 备注  
CPU core i7-4510U 内存 ：4GB  
操作系统：win10

## CPU指令集

基本指令集共有47条指令。

实现16条指令：LUI, AUIPC, ADDI, ANDI, ORI, XORI, LBU, SLTI, SRAI, JAL, SB, AND, OR, XOR, ADD, SUB

## 程序框架

实现的指令中：

1. LUI存放立即数到rd的高20位，低12位置0；
2. AUIPC将立即数与PC地址相加存放结果至rd的高20位，低20置0；
3. ADDI将立即数与rs1相加的结果存放rd中；
4. ANDI将立即数与rs1逻辑与的结果存放rd中；
5. ORI将立即数与rs1逻辑或的结果存放rd中；
6. XORI将立即数与rs1逻辑异或的结果存放rd中；
7. LBU从内存中加载8位无符号数存放进rd中；
8. SLTI，如果rs1比立即数小将1放入rd中，否则放0；
9. SRAI根据立即数将rs1进行算数右移，结果存放于rd中；
10. JAL将PC地址累加4存放入rd中；
11. SB将存储器的低8位存入存储器中；
12. AND将rs2与rs1的逻辑与的结果放入rd中；
13. OR将rs2与rs1的逻辑或的结果放入rd中；
14. XOR将rs2与rs1的逻辑异或的结果放入rd中；
15. AND将rs2与rs1的和的结果放入rd中；
16. SUB将rs2与rs1差的结果存放rd中；

头文件与库文件声明如下

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

入参定义如下，包括了输入输出、时钟、重置等信号。

entity 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\_in: in std\_logic\_vector(31 downto 0);

data\_out: out std\_logic\_vector(31 downto 0);

data\_read: out std\_logic;

data\_write: out std\_logic

);

end entity cpu\_singlecycle;

结构部分，声明了需要使用的变量信号。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funct7，5位的偏移量，这四个变量由读取ir的指令译码获得，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\_12U : std\_logic\_vector(31 downto 0);

signal Imm11\_0I : std\_logic\_vector(31 downto 0);

signal Imm20\_1J : std\_logic\_vector(31 downto 0);

signal Imm12\_1B : std\_logic\_vector(31 downto 0);

signal Imm11\_0S : std\_logic\_vector(31 downto 0);

signal src1: std\_logic\_vector(31 downto 0);

signal src2: std\_logic\_vector(31 downto 0);

signal sb\_a1: std\_logic\_vector(31 downto 0);

signal sb\_d1: std\_logic\_vector(31 downto 0);

signal LUIresult: std\_logic\_vector(31 downto 0);

signal AUIPCresult: std\_logic\_vector(31 downto 0);

signal ADDIresult: std\_logic\_vector(31 downto 0);

signal ANDIresult: std\_logic\_vector(31 downto 0);

signal ORIresult: std\_logic\_vector(31 downto 0);

signal XORIresult: std\_logic\_vector(31 downto 0);

signal LBUresult: std\_logic\_vector(31 downto 0);

signal SLTIresult: std\_logic\_vector(31 downto 0);

signal SRAIresult: std\_logic\_vector(31 downto 0);

signal JALresult: std\_logic\_vector(31 downto 0);

signal SBresult: std\_logic\_vector(31 downto 0);

signal ANDresult: std\_logic\_vector(31 downto 0);

signal ORresult: std\_logic\_vector(31 downto 0);

signal XORresult: 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);

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);

指令译码过程，获取各个信号相应的值：

inst\_addr <= pc;

ir <= inst;

-- Decode

-- Not finished

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\_12U <= ir(31 downto 12) & "000000000000";

Imm11\_0I <= "11111111111111111111" & ir(31 downto 20) when ir(31)='1' else

"00000000000000000000" & ir(31 downto 20);

Imm20\_1J <= "111111111111" & ir(31) & ir(19 downto 12) & ir(20) & ir(30 downto 21) when ir(31)='1' else

"000000000000" & ir(31) & ir(19 downto 12) & ir(20) & ir(30 downto 21);

Imm11\_0S <= "11111111111111111111" & ir(31 downto 25) & ir(11 downto 7) when ir(31)='1' else

"00000000000000000000" & ir(31 downto 25) & ir(11 downto 7);

Imm12\_1B <= "11111111111111111111" & ir(31) & ir(7) & ir(30 downto 25) & ir(11 downto 8) when ir(31)='1' else

"00000000000000000000" & ir(31) & ir(7) & ir(30 downto 25) & ir(11 downto 8);

-- Read operands from register file

src1 <= regs(TO\_INTEGER(UNSIGNED(rs1)));

src2 <= regs(TO\_INTEGER(UNSIGNED(rs2)));

sb\_d1 <= src2 and "11111111";

sb\_a1 <= STD\_LOGIC\_VECTOR (SIGNED(src1) + SIGNED(Imm11\_0S));

-- Prepare index and data to write into register file

reg\_write\_id <= rd;

执行阶段，根据指令译码获取到的值，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。

ADDIresult <= STD\_LOGIC\_VECTOR(SIGNED(src1) + SIGNED(Imm11\_0I));

SLTIresult <= "00000000000000000000000000000001" when TO\_INTEGER(UNSIGNED(src1)) < TO\_INTEGER(UNSIGNED(Imm11\_0I)) else

"00000000000000000000000000000000";

ANDIresult <= src1 and Imm11\_0I;

ORIresult <= src1 or Imm11\_0I;

XORIresult <= src1 xor Imm11\_0I;

LUIresult <= Imm31\_12U;

AUIPCresult <= STD\_LOGIC\_VECTOR(SIGNED(pc) + SIGNED(Imm31\_12U));

ADDresult <= STD\_LOGIC\_VECTOR(SIGNED(src1) + SIGNED(src2));

SUBresult <= STD\_LOGIC\_VECTOR(SIGNED(src1) - SIGNED(src2));

ANDresult <= src1 and src2;

ORresult <= src1 or src2;

XORresult <= src1 xor src2;

SRAIresult <= to\_stdlogicvector( to\_bitvector(src1) SRA to\_integer(unsigned(shamt)) ) ;

LBUresult <= "000000000000000000000000" & data\_in(7 downto 0);

JALresult <= STD\_LOGIC\_VECTOR(UNSIGNED(pc)+4);

reg\_write\_data <= ADDIresult when opcode = "0010011" and funct3 = "000" else

SLTIresult when opcode = "0010011" and funct3 = "010" else

XORIresult when opcode = "0010011" and funct3 = "100" else

ORIresult when opcode = "0010011" and funct3 = "110" else

ANDIresult when opcode = "0010011" and funct3 = "111" else

ANDresult when opcode = "0110011" and funct3 = "111" else

ORresult when opcode = "0110011" and funct3 = "110" else

XORresult when opcode = "0110011" and funct3 = "100" else

ADDresult when opcode = "0110011" and funct7 = "0000000" else

SUBresult when opcode = "0110011" and funct7 = "0100000" else

SRAIresult when opcode = "0010011" and funct7 = "0100000" else

LBUresult when opcode = "0000011" and funct3 = "100" else

JALresult when opcode = "1101111" else

"00000000000000000000000000000000";

data\_addr <= sb\_a1 when opcode = "0100011" and funct3 = "000";

data\_out <= sb\_d1 when opcode = "0100011" and funct3 = "000";

next\_pc <= STD\_LOGIC\_VECTOR(UNSIGNED(pc) + UNSIGNED(Imm20\_1J)) when opcode = "1101111" else

STD\_LOGIC\_VECTOR(UNSIGNED(pc) + UNSIGNED(Imm12\_1B)) when opcode = "1100011" and funct3 = "110" and (SIGNED(src1) < SIGNED(src2)) else

STD\_LOGIC\_VECTOR(UNSIGNED(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

## 分析和结论

从测试记录来看，模拟器实现了对二进制指令文件的读入，指令功能的模拟，CPU和存储器状态的输出。 根据结果，可以认为编写的模拟器实现了所要求的功能，完成了实验目标。

## 心得体会

我原本计划在原来逻辑与电路课程实验做的CPU基础上改善完善设计，但涉及的RISCV基本指令有不同格式需要对译码器进行大量修改，我只能在原来的设计上选择几条简单的指令实现，分模块实现各个部件功能之后再建立数据通路的设计难度比较大，因此我选择了使用硬件语言描述了单周期CPU模拟器。这次只实现了16条指令，还有一大半指令没有实现，由于很多指令结构和操作相似因此省略没有实现，遗憾的是没有能够实现计算机组成与设计课程讲过的流水线CPU，建立流水线CPU的数据通路需要深入复习了解原先的内容，希望在今后的学习过程中，有机会能具体实现这部分内容。