实验报告

班级：智能 1602
姓名：李路
学号：201608010623

实验内容

实现单周期 CPU 的设计。

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

模拟环境

部件 配置 备注
CPU core i7 内存 ： 8GB
操作系统：win10 专业版

CPU 指令集

基本指令集共有 47 条指令。

RV32I Base Instruction Set

imm[31:12]			rd	0110111	LUI
imm[31:12]			rd	0010111	AUIPC
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

程序框架

程序被分为三个部分。取指 译码 执行

一是对输入输出信号、寄存器变量的定义与初始化，二是 获取寄存器变量之后进行指令相应的计算与赋值，最后是写回操作。 JAL、BLTU、SB、XORI、ADD 指令的作用分别如下：

1、JAL：直接跳转指令，并带有链接功能，指令的跳转地址在指令中，跳转发生 时要把返回地址存放在 R[rd]寄存器中。

2、BLTU：为无符号比较，当 R[rs1]<R[rs2]时,进行跳转。

3、SB：SB 指令取寄存器 R[rs2]的低位存储 8 位值到存储器。有效的字节地址是 通过将寄存器 R[rs1]添加到符号扩展的 12 位偏移来获得的。

4、XORI:在寄存器 R[rs1]上执行位 XOR 的逻辑操作，并立即将符号扩展 12 位，将 结果放

在 R[rd]中。注意：XORIR[rd]，R[rs1]，-1 执行寄存器 R[rs1]的位逻辑反转。
5、ADD:进行加法运算，R[rs1]+R[rs2]，将结果存入 R[rd]中。 输入参定义如下，包括了输入输出、时钟、重置等信号。

五条指令中，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。LBH 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。

```
entity cpu 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;
```

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);
```

获取对于指令要求的立即数的值：

```
    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;
```

当程序执行到一定步骤，将结果保存下来

```
        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);
        -- more
        -- ......

        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
                          -- more
                          -- ......
                          -- At last, set a default value
                          "00000000000000000000000000000000";

        -- Execute
        -- Not finished

        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);
```

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 <= 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
                      -- more
                      -- ......
                      -- At last, set a default value
                      "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);
```

最后写回阶段，当时钟上跳时触发。

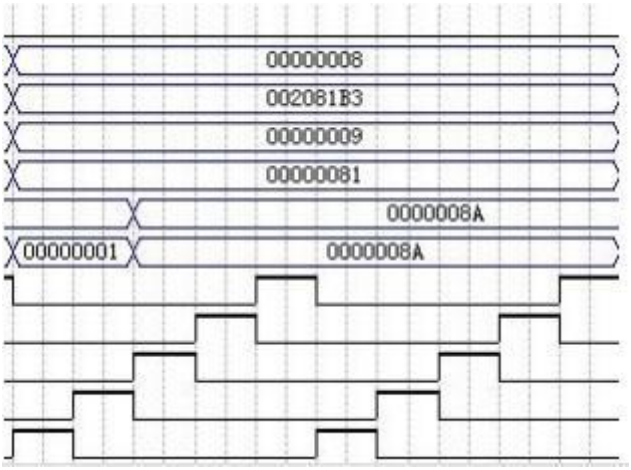
```
-- Update pc and register file at rising edge of clk
process(clk)
begin
    if(rising_edge(clk)) then
        if (reset='1') then
            pc <= "00000000000000000000000000000000";
            -- Clear register file?
        else
            pc <= next_pc;

            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
```

测试结果:

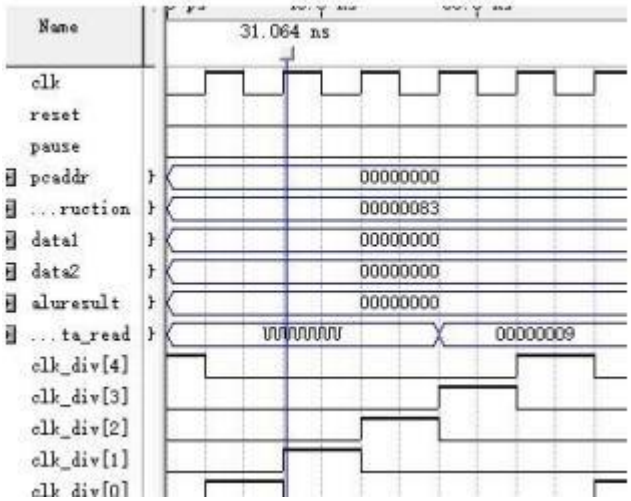
ADD 指令

00000000001000001000000110110011



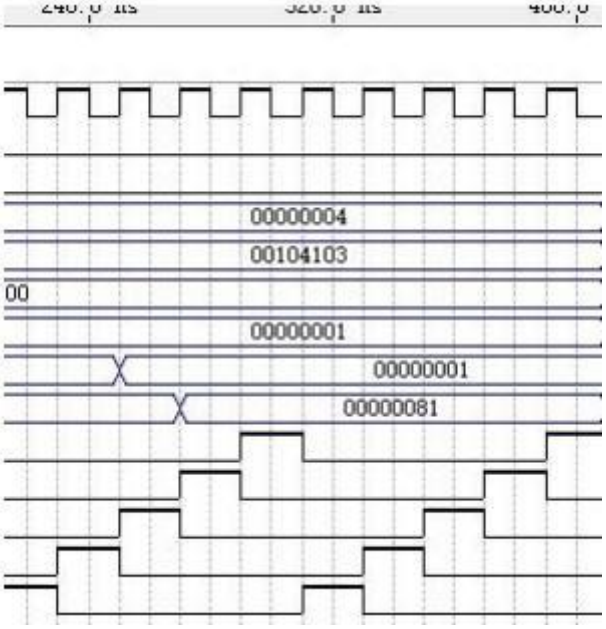
LB 指令

00000000000000000000000010000011



LBU 指令

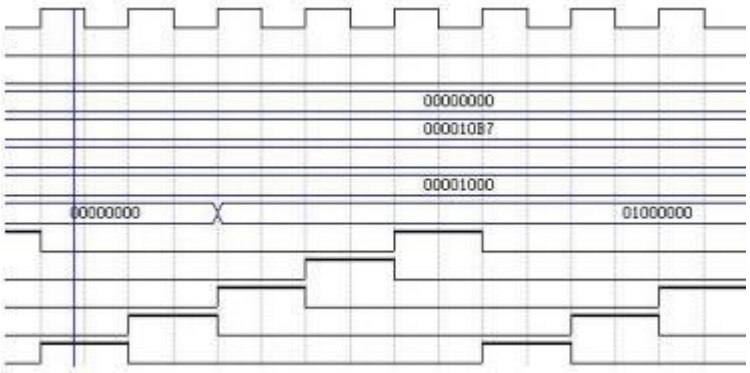
00000000000010000010000010000011



LUI x1，0x1

00000000000000000001000010110111

r1=00000000000000000001000000000000



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。

根据分析结果，可以认为编写的模拟器实现了所要求的功能，完成了几条指令的实验目标，剩余指令有待完善。

心得体会

通过本次实验，使得更加了解了 CPU 的整个工作流程，我也更加细致地知道了 CPU 中微指令程序的运行过程，原本想分模块实现各个部件功能之后，再将其整合实现 CPU。但是由于能力有限未能完成，不过还是已实现的部分模块的功能。在今后的学习过程中， 我希望能将这部分的能力进一步提升。CPU 设计还是比较难的，需要花很多的精力和时间在上面，但是这一切都很值得，希望后面有时间能够进行完善，虽然没有完成整体设计，但还是了解了很多相关的知识，对整个 CPU 设计的把握还是有的，小学期还是挺有收获的。