湖南大學

HUNAN UNIVERSITY

课程实验报告

课程名称：微处理器设计

学生姓名：李叙庆

学生学号：201608010515

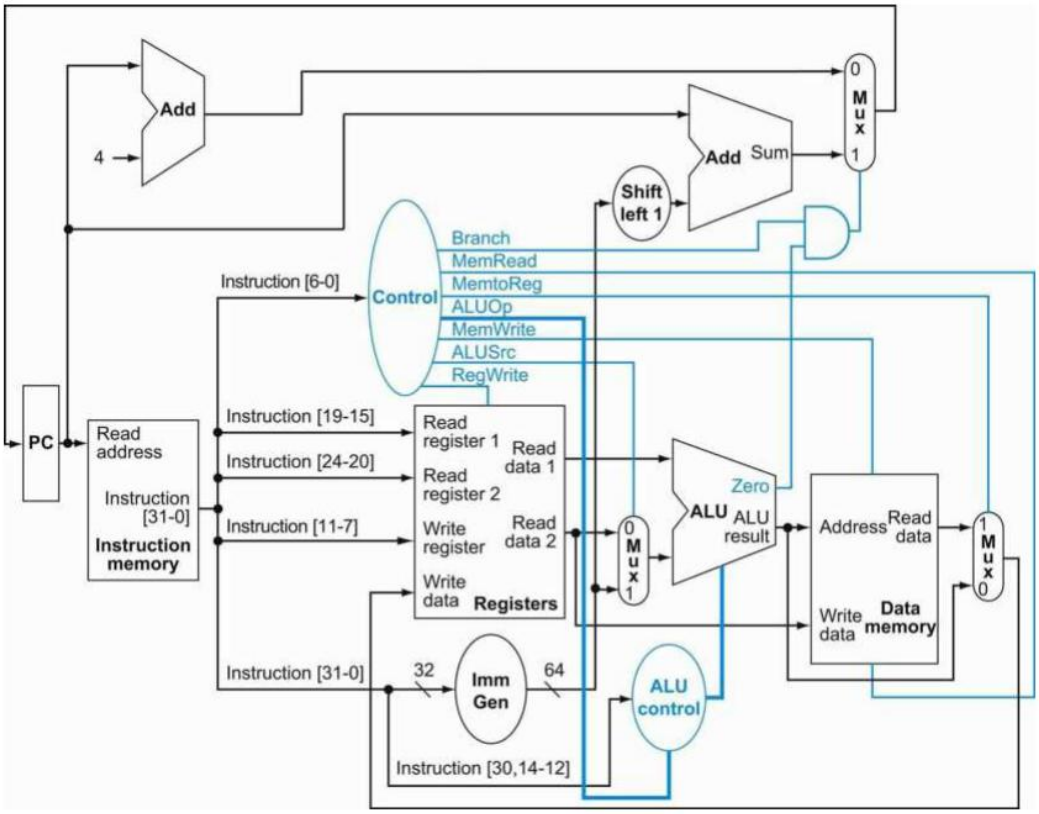
专业班级：计科 1605

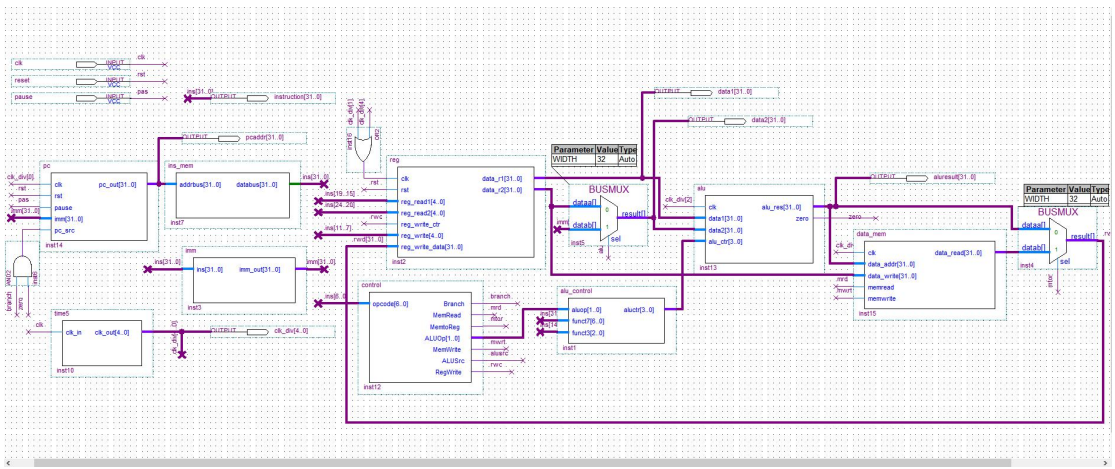
实验任务

完成一个执行 RISC-V 的基本整数指令集 RV32I 的 CPU 设计（单周期实现）

实验流程

一、参考设计手册给出的 datapath，连好的整体框架。



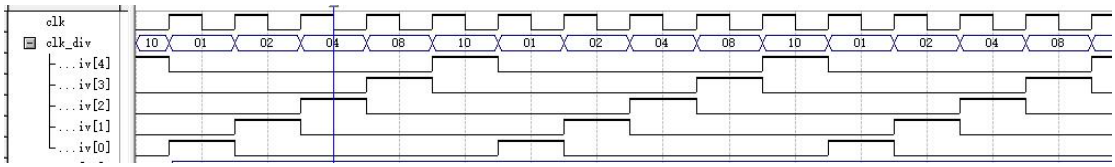


五个模块：

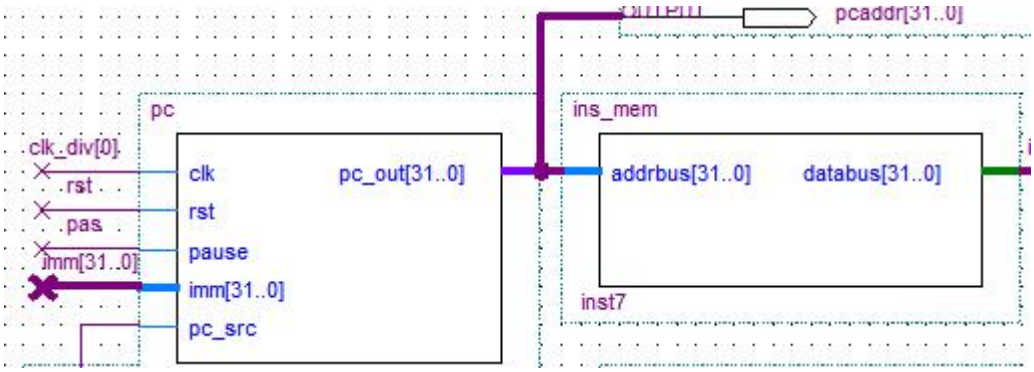
Clk5 分时模块：按照手册的 5 级流水线设计单周期 cpu，把 clk 分为 5 分时。每 5 个 clk 循环执行：取指令，译码，执行，访存，写回。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity time5 is
port(
    clk_in:in std_logic;
    clk_out:out std_logic_vector(4 downto 0)
    );
end time5;

architecture bhv of time5 is
signal temp:std_logic_vector(4 downto 0):="10000";
begin
    t0:process(clk_in)
    begin
        if(rising_edge(clk_in)) then
            if temp="10000" then
                temp<="00001";
            else
                temp <= temp(3 downto 0) & '0';
            end if;
        end if;
    end process;
    clk_out <= temp;
end ;
```

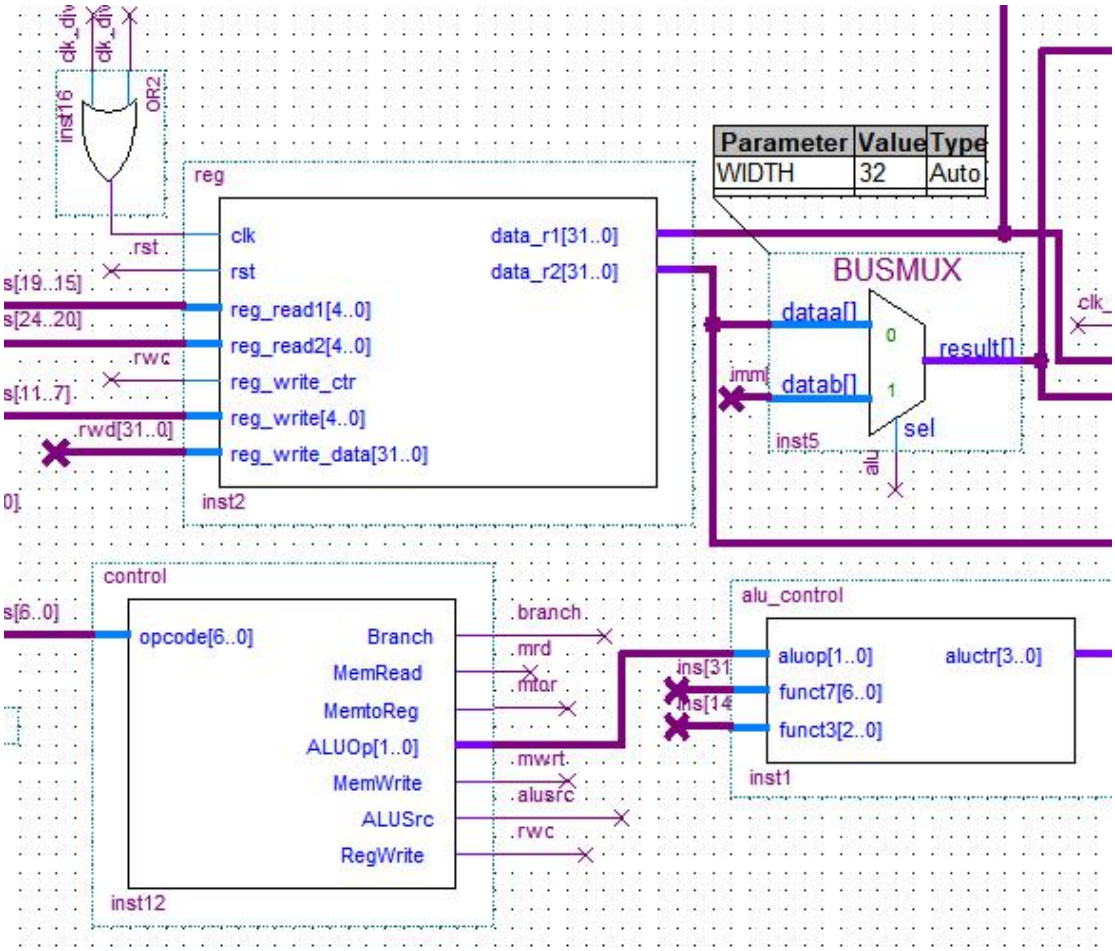
仿真结果



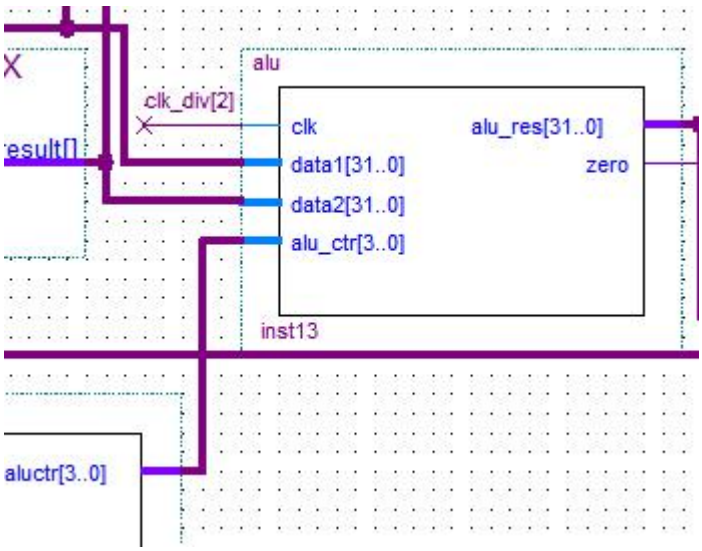
取指阶段 clk1：产生一条 pc 输出，pc 进入指令存储器（ins_mem）取出指令。



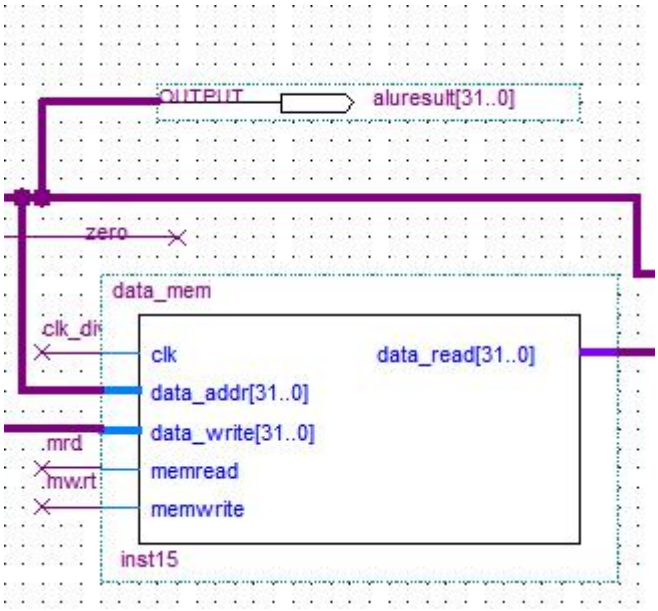
译码阶段 clk2：
其实在指令取出来后，指令进入 2 个部分：
一个是 imm（立即数产生器）根据指令类型生成相应立即数。
另一个，同时指令还进入 control（控制器）产生控制信号：
Branch：branch=1 则下一条 pc 地址要跳转，branch=0 则下一条 pc 选择 pc+4.
MemRead：等于 1 则是要从数据存储器读取数据（load）
MemtoReg：等于 1 则是选择从存储器出来的数据写入到寄存器，0 则是 alu 计算的结果
ALUOp：进入 alu_control 器件，与 funct7 和 funct3 一同控制 alu 运算规则。
MemWrite：数据存储器写入数据控制，1 允许写入
ALUSec：进入 alu 的数据来源，等于 0 选择寄存器出来的数，等于 1 选择产生的立即数。
RegWrite：寄存器写入控制信号，1 写入有效



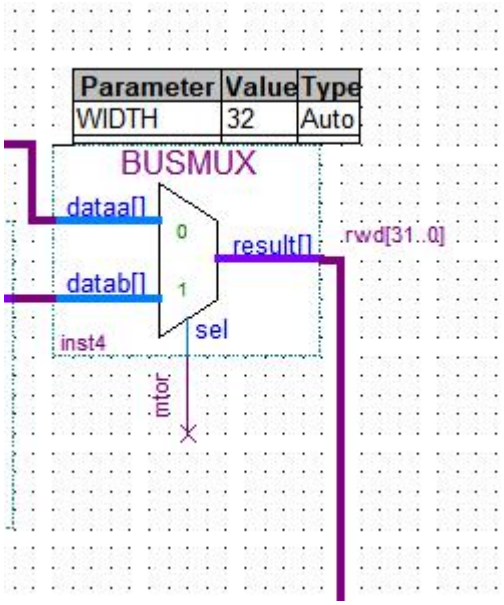
执行阶段（clk3 上升沿产生 alu 结果输出）：



访问存储器阶段：



写回阶段：



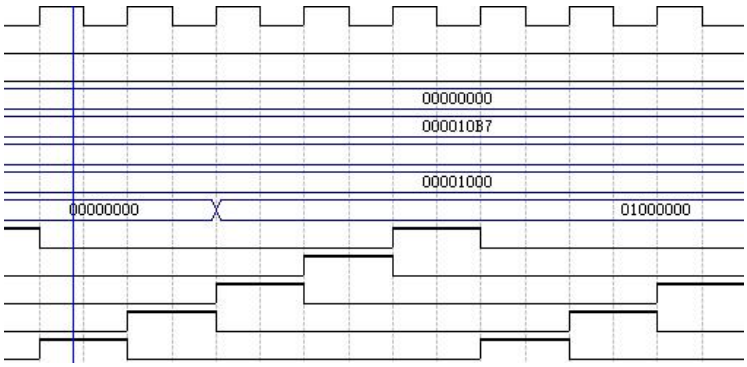
测试样例：

LUI x1，0x1

00000000 00000000 0001 00001 0110111

r1=00000000 0000000 00010000 00000000

测试结果



LB

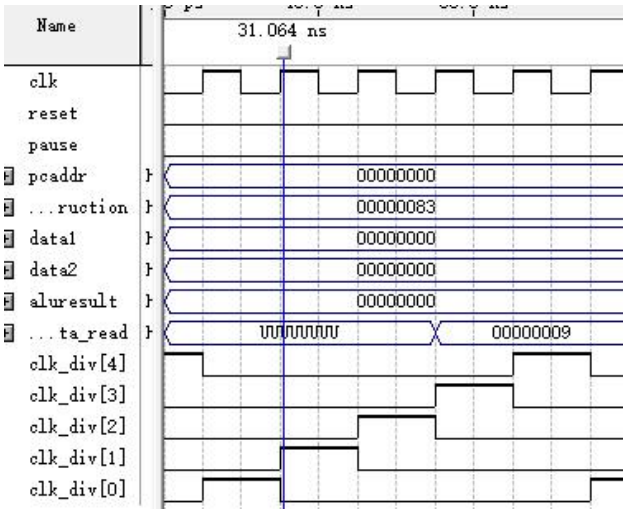
000000000000 00000 000 00001 0000011

Imm rs1 fc3 rd op

加载存储位置 0 的 8bit 有符号数到寄存器 1 中

```
type memtype is array(natural range<>)
signal memdata: memtype(1023 downto 0)
    0 => X"09",
    1 => X"81",
    2 => X"01",
    3 => X"80",
```

结果是 9



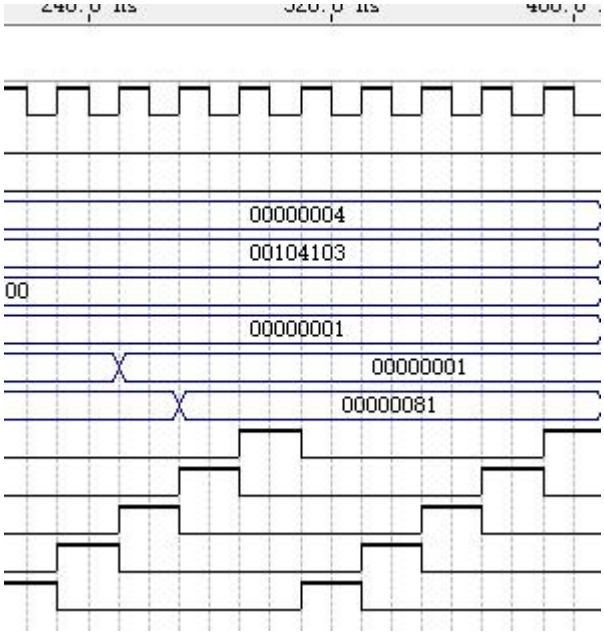
LBU rs1 funct3 rd op

000000000001 00000 100 00010 0000011

Imm rs1 funct3 rd op

0+1=0 地址读取 8bit 无符号 dao 寄存器 2

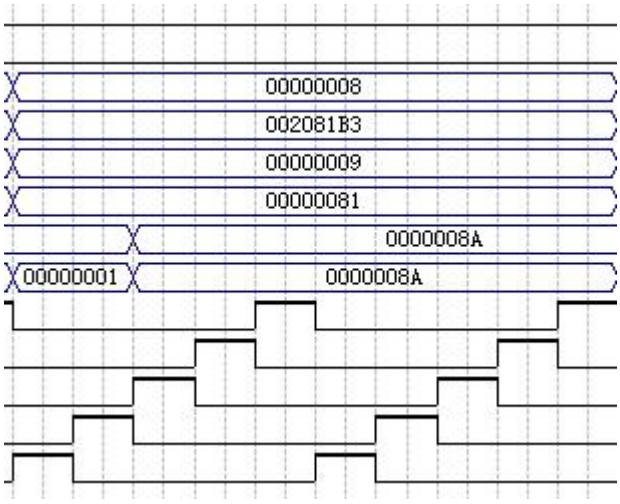
```
type memtype is array(natural range<>)
signal memdata: memtype(1023 downto 0)
    0 => X"09",
    1 => X"81",
    2 => X"01",
    3 => X"80",
```



ADD 指令

0000000 00010 00001 000 00011 0110011

rs1 与 rs2 相加，存到寄存器 x3

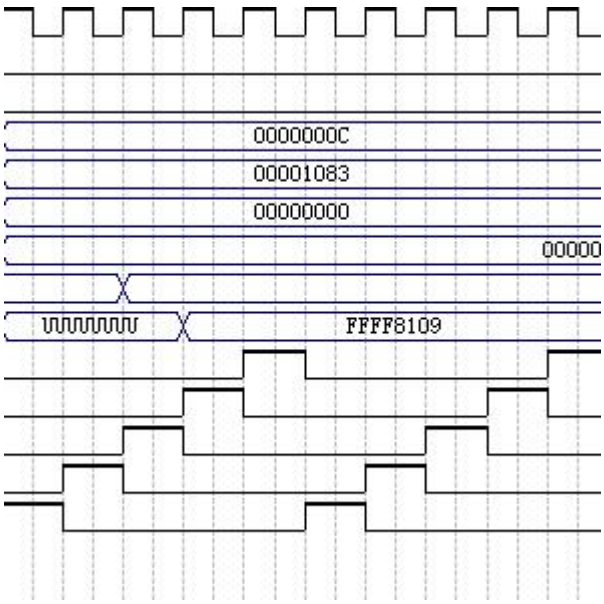


有符号扩展的例子：

LH 16bit signed

000000000000 00000 001 00001 0000011

Imm rs1 fc3 rd op



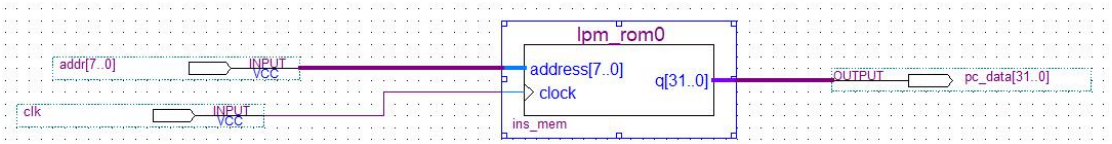
可见上述指令基本正确：

实验总结：

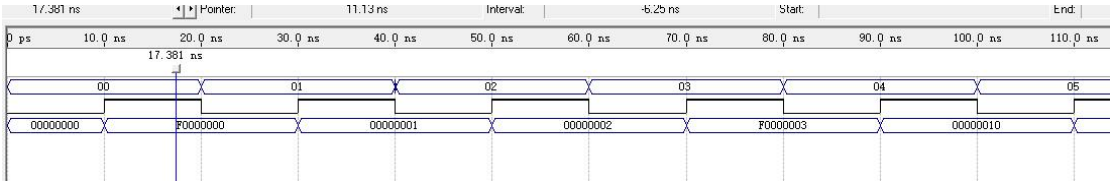
这次小学期有四个星期。

在第一个星期，我基本都是在看指令，觉得好多内容，又一时难以理解。同时看了老师的 sample 代码有很多名字看不懂，第一周基本是去读这些内容。

第二个星期，开始翻译手册的内容，慢慢读懂了他的 datapath 建立过程，以及各个模块间的联系。然后似乎想起一句话“一切从指令出发”，我就从指令模块开始写，库里的 lpm_rom 模块的地址宽度超过 8 位后编译报错，可能这么大地址空间的仿真太消耗资源。然后我按老师代码用 vhdl 写了一个 32 位的模块（其实不需要，就用 lpm_ram 模块，然后地址线低 8 位连上就可以了，见下图）。然后写 pc 模块，慢慢找到自信。这周基本完成一半的 datapath



Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	F0000000	00000001	00000002	F0000003	00000010	00000011	00000012	00000013
8	00000020	00000021	00000022	00000023	00000030	00000031	00000032	00000033
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000



第三、四周，经过之前的积累，对指令更加熟悉了。开始进行最重要的 alu 模块和指令控制模块的编写。这里的妙处在把指令用到相同逻辑运算产生相同的控制信号，比如，add 和 addi，alu 的控制信号相同都是进行加法操作，而不同在 alu 接受的数据输入不同，add 是 2 个来自寄存器的，addi 有一个来自立即数模块根据指令格式事先生成好的。Alu 只管计算就好。

最终勉强完成，约 37 指令理论上的译码执行都写好了，测试没有全部进行，可留个学弟学妹解决嘻嘻。