目录

完成内容

1. ALU 运算子电路（练习 1）

1.1 设计思路

在本次设计中，ALU 总共包括 15 种运算。在设计过程中，将每种运算所得结果连接到相应的“隧道”上，如 add0，and1，or2……再通过一个 16 路复用器，结合 ALUSel 来选择对应得运算结果。

其中 add，and，or，xor，srl，sra，sll，slt，divu，remu，mul，mulhu，sub，bsel 均可通过简单的逻辑电路组合来实现，而 mulh 则相对比较复杂，下面重点说明 mulh 的实现过程与思路，具体电路设计见 1.2。

首先，通过对 Multiplier 的实验，可知道 

右边所输出的是做有符号数运算时的低 32 位，下面 carry out 输出的是做无符号数运算时的高 32 位，分别对应 mul 和 mulhu。

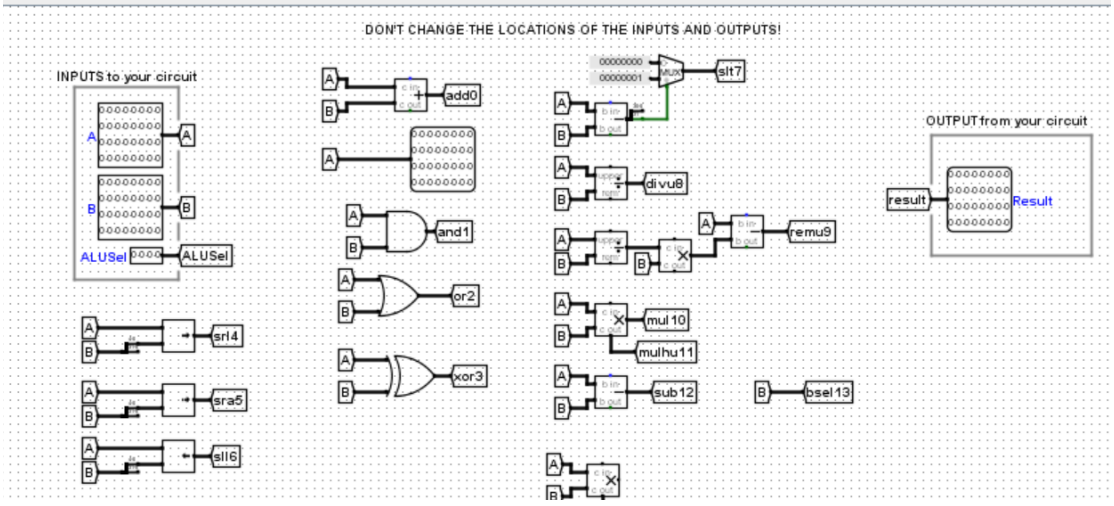
而对于 mulh，首先需要判断相乘的两个数的正负（补码表示的 32 位数，若最高位为 1，该数为负数；若最高位为 0，该数为正数）。**如果存在一个负数一个正数，**则将负数再进行一个“补码”还原成正数，然后两数还原后的正数相乘，相乘得到的低 32 为需要进行“补码”，同时，若低 32 位的最高位为 1 且低 32 位“补码“后最高位为 0，说明要向前进一位，所以此时高 32 位就要进行“补码”，即反码后再加一；否则，高 32 位只需取反码即可。

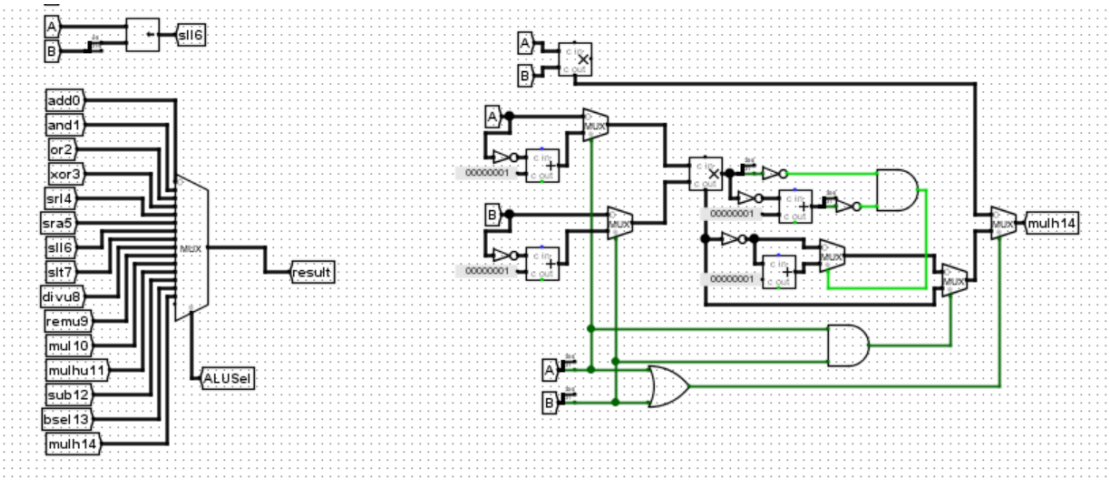
如果两个数都是负数，负负得正。同样将俩负数都“补码”还原成正数，mulh 就是这俩正数进行乘运算后 carry out 所输出的。

如果两个数都是正数，直接取这俩正数相乘后 carry out 所输出的即可。

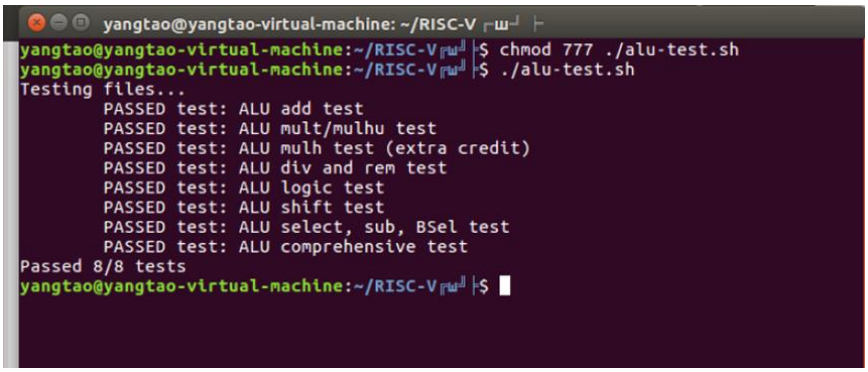
1.2 具体设计内容

ALU 子电路详细设计内容如下：





1.3 测试结果



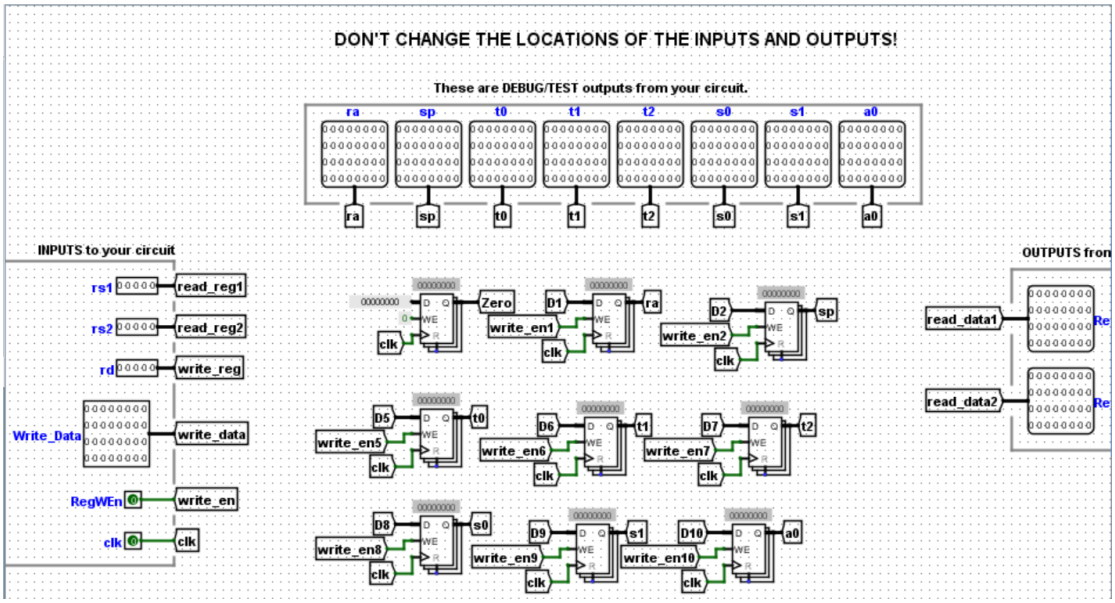
2. RegFile 寄存器文件子电路（练习 2）

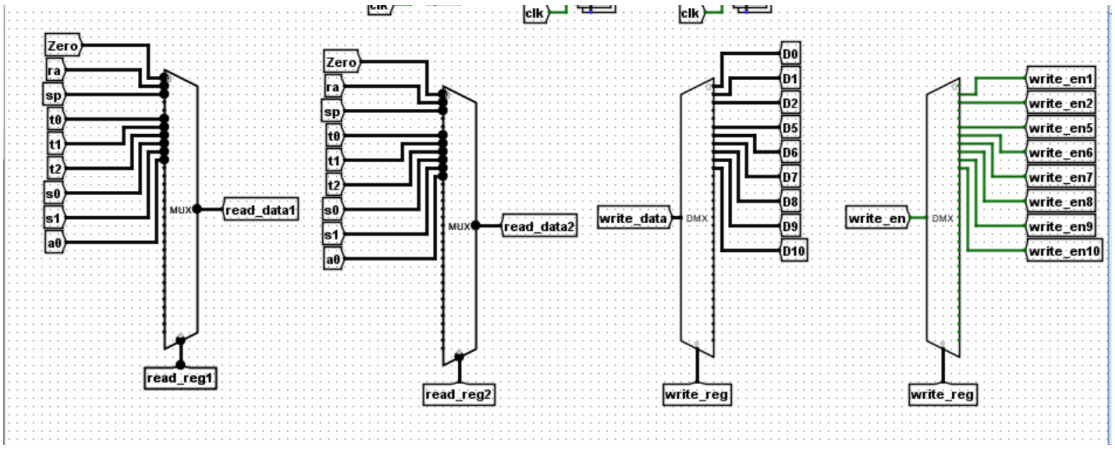
2.1 设计思路

首先，添加 9 个 register，其中 0 号寄存器因为总是 0，所以写使能设为常数 0，其他 8 个寄存器各自的写使能为 write_enx(x 为寄存器编号)，写入数据为 Dx（x 为寄存器编号）。然后，利用 32 路 MUX 和 read_reg1 和 read_reg2 选择出要读取的寄存器并得到 read_data1 和 read_data2。

然后，利用 32 路 DMX 和 write_reg 确定 Dx 和 write_enx 的值。将 write_data 写入 write_reg 号寄存器中，将 write_en 赋值给 write_reg 号寄存器的 write_enx。这样做是为了避免写某一个寄存器的时候其他寄存器收到影响。

2.2 具体设计内容





2.3 测试结果

```
Passed 8/8 tests
yangtao@yangtao-virtual-machine:~/RISC-V$ chmod 777 ./reg-test.sh
yangtao@yangtao-virtual-machine:~/RISC-V$ ./reg-test.sh
Testing files...
        PASSED test: RegFile insert test
        PASSED test: RegFile zero test
        PASSED test: RegFile x0 test
        PASSED test: RegFile all registers test
Passed 4/4 tests
yangtao@yangtao-virtual-machine:~/RISC-V$
```

3. addi 指令（练习 3）

3.1 设计思路

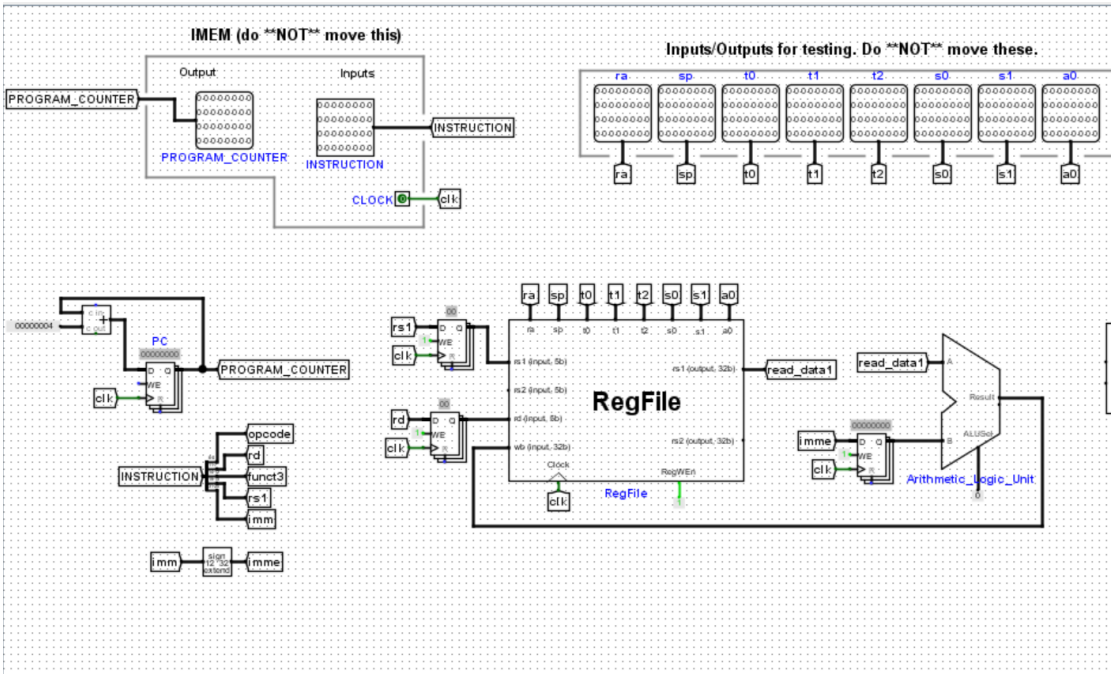
首先，根据 addi 指令格式将指令解码成 opcode，rd，funct3，rs1，imm 五部分。

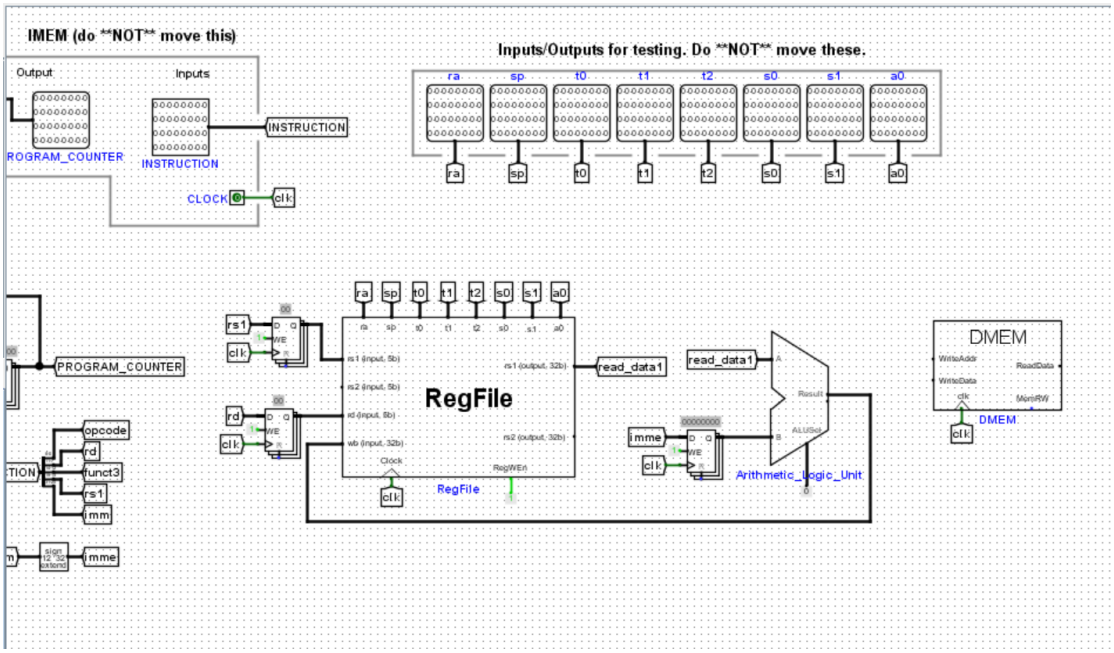
imm[11:0]	rs1	funct3	rd	opcode	I-type

并通过一个位拓展器将 12 位的 imm 符号扩展为 32 位的 imme。将 rs1，rd 连接到 Regfile 对应的位置上。

对于 addi 指令，将 rs1 寄存器内容与 32 位立即数 imme 相加后存到 rd 号寄存器中。在执行 addi 指令时，各控制信号可设置为常数。RegWr 设为 1，ALUsel 设为 0，相应的位置连接上时钟 clk 即可实现 addi。

3.2 具体设计内容





3.3 测试结果

```
yangtao@yangtao-virtual-machine:~/RISC-V$ ./cpu-part1-sing
Testing submission
        ADDI PASSED test: CPU addi test
        Score for ADDI: 1/1 (1/1 tests passed, 0 partially)
ADDI: 1/1 (1/1 tests passed, 0 partially)
        Passed test "CPU addi test" worth 1 points.
yangtao@yangtao-virtual-machine:~/RISC-V$
```

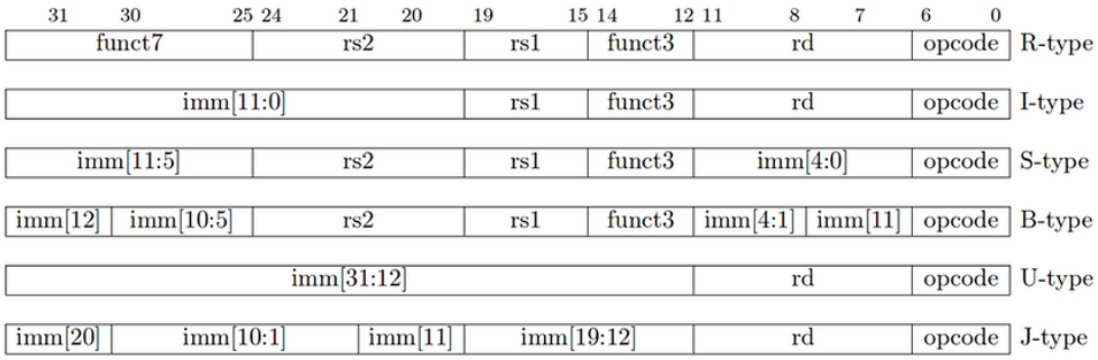
```
yangtao@yangtao-virtual-machine:~/RISC-V$ ./cpu-part1-pipelined.sh
Testing submission
        ADDI PASSED test: CPU addi test
        Score for ADDI: 1/1 (1/1 tests passed, 0 partially)
ADDI: 1/1 (1/1 tests passed, 0 partially)
        Passed test "CPU addi test" worth 1 points.
```

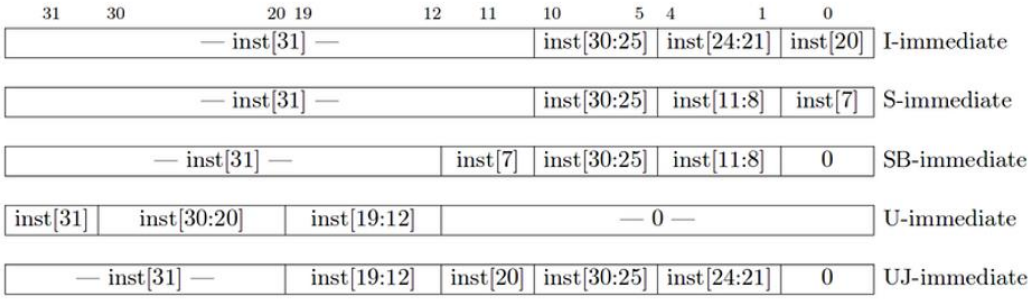
4. CPU 数据通道及控制器，立即数生成器

4.1 立即数生成器

4.1.1 设计思路

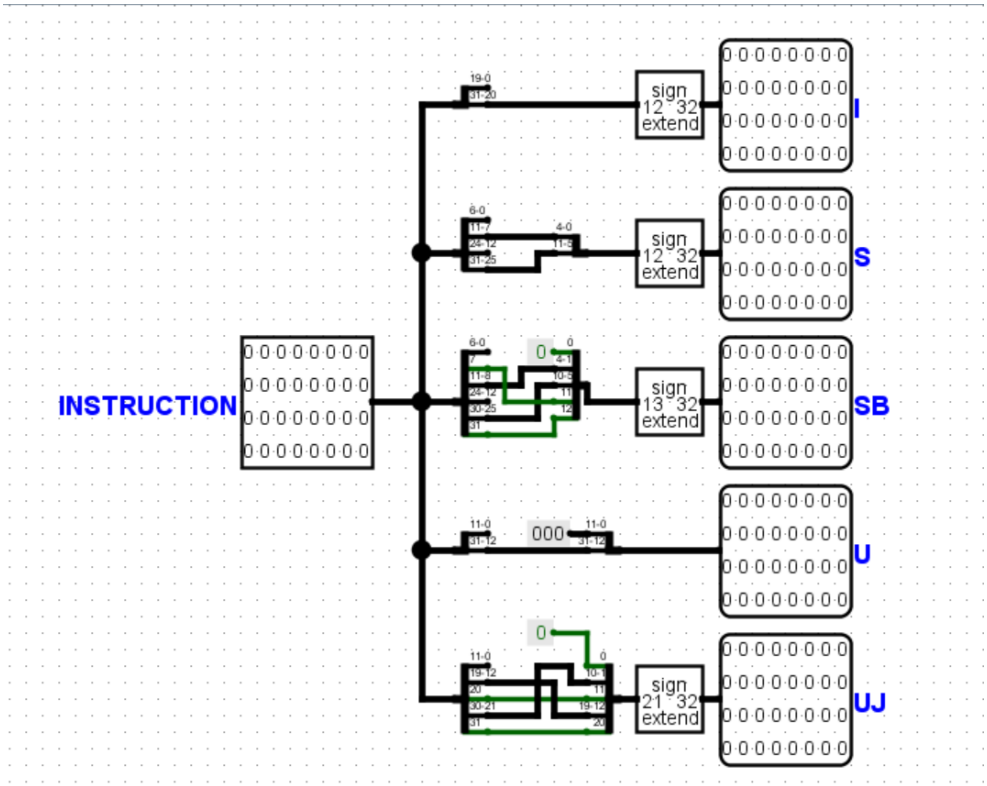
根据第二部分提供材料中指令集格式（如图），以及各类指令立即数格式可得出各指令中的立即数。取出指令中相应位的值后再进行符号扩展即可。





4.1.2 具体设计内容

立即数生成器详细设计如下：



4.2 控制器

4.2.1 设计思路

控制信号有 ALUsrc（立即数是否进入 ALU 进行运算），MemtoReg（是否从内存读取数据写入到寄存器中），RegWr（是否写寄存器），MemWr（是否写内存），nPCsel（是否是跳转指令），ExtOp（选择对应类型指令的立即数），ALUctr(ALU 中进行的运算类型)，Jump（是否是 jal 或者 jalr 指令），LoadOp（装入的字节数）。控制信号由 opcode，funct3，funct7 决定。

列出真值表如下：

指令	Add	Mul	Sub	Sll	Mulh	Mulhu	Slt	Xor
ALUsrc	0	0	0	0	0	0	0	0
MemtoReg	0	0	0	0	0	0	0	0
RegWr	1	1	1	1	1	1	1	1
MemWr	0	0	0	0	0	0	0	0
nPCsel	0	0	0	0	0	0	0	0
ExtOp	X	X	X	X	X	X	X	X
ALUctr	0000	1010	1100	0110	1110	1011	0111	0011
Jump	0	0	0	0	0	0	0	0
LoadOp	00	00	00	00	00	00	00	00

指令	Divu	Srl	Or	Remu	And	Lb	Lh	Lw
ALUsrc	0	0	0	0	0	1	1	1
MemtoReg	0	0	0	0	0	1	1	1
RegWr	1	1	1	1	1	1	1	1
MemWr	0	0	0	0	0	0	0	0
nPCsel	0	0	0	0	0	0	0	0
ExtOp	X	X	X	X	X	000	000	000
ALUctr	1000	0100	0010	1001	0001	0000	0000	0000
Jump	0	0	0	0	0	0	0	0
LoadOp	00	00	00	00	00	10	01	00

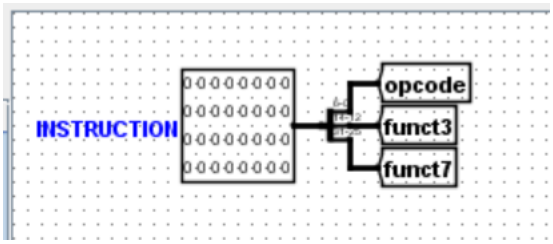
指令	Addi	Slli	Slti	Xori	Srli	Srai	Ori	Andi
ALUsrc	1	1	1	1	1	1	1	1
MemtoReg	0	0	0	0	0	0	0	0
RegWr	1	1	1	1	1	1	1	1
MemWr	0	0	0	0	0	0	0	0
nPCsel	0	0	0	0	0	0	0	0
ExtOp	000	000	000	000	000	000	000	000
ALUctr	0000	0110	0111	0011	0100	0101	0010	0001

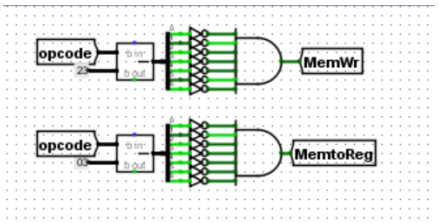
Jump	0	0	0	0	0	0	0	
LoadOp	00	00	00	00	00	00	00	00

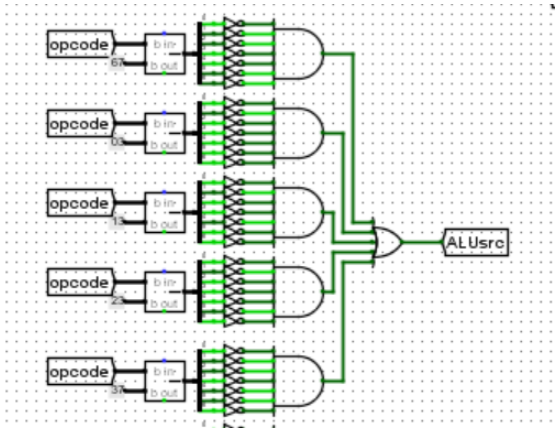
指令	Jalr	Sw	Beq	blt	Bltu	bne	Lui	jal
ALUsrc	1	1	0	0	0	0	1	0
MemtoReg	0	0	0	0	0	0	0	X
RegWr	1	0	0	0	0	0	1	1
MemWr	0	1	0	0	0	0	0	0
nPCsel	0	0	1	1	1	1	0	X
ExtOp	000	001	010	010	010	010	011	100
ALUctr	0000	0000	1100	0111	1000	1100	1101	0000
Jump	1	0	0	0	0	0	0	1
LoadOp	00	X	X	X	x	X	00	00

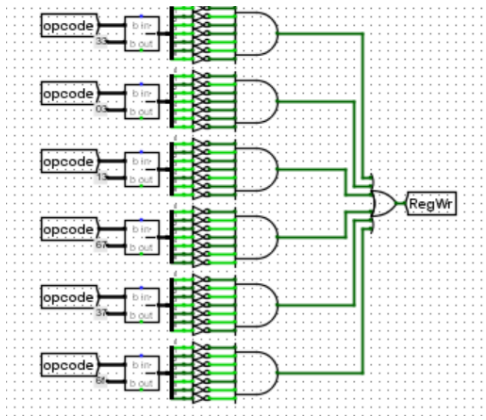
4.2.2 具体设计内容

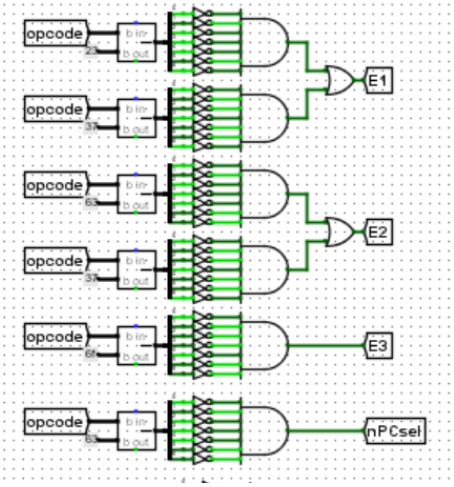
根据以上真值表画出相应的电路如下：

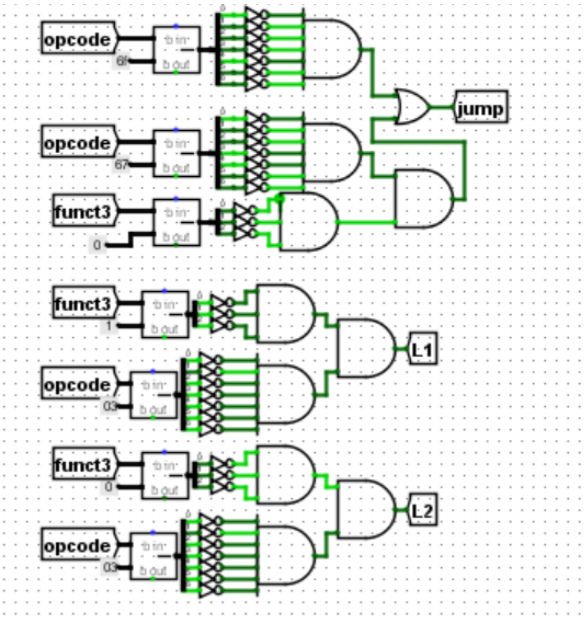


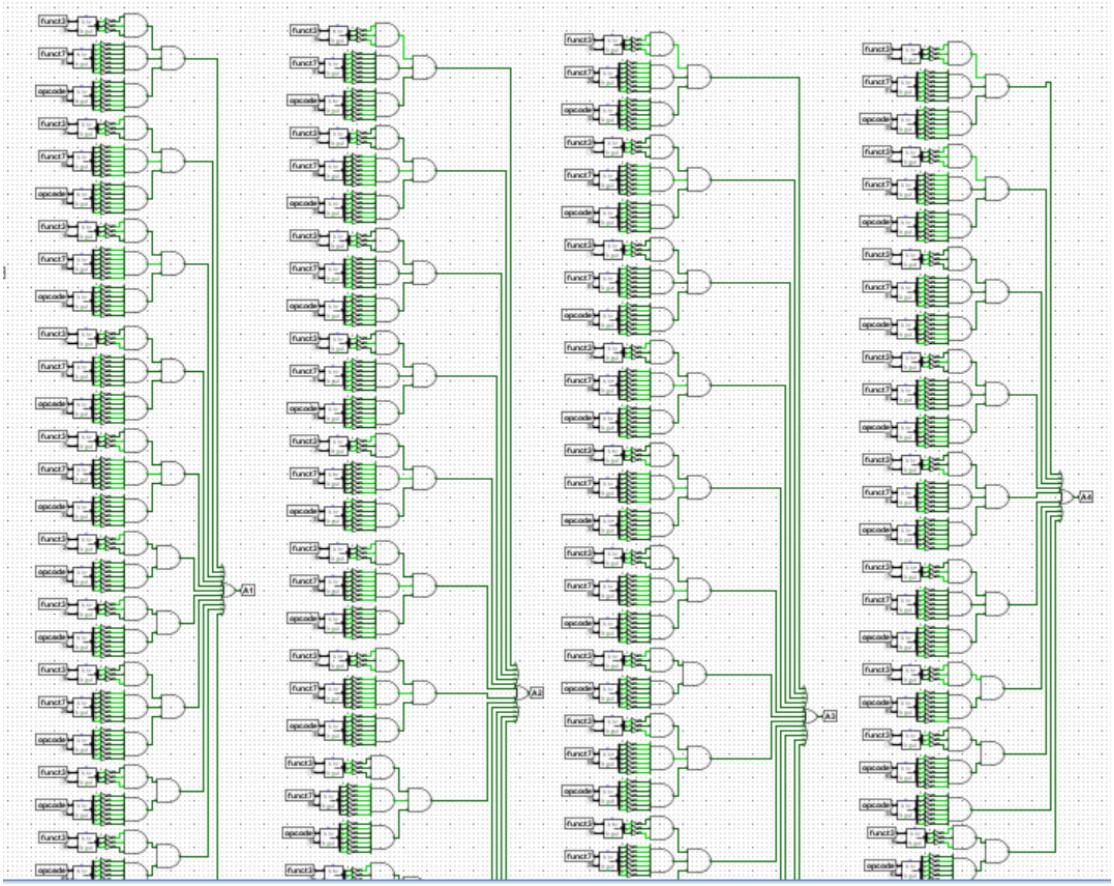


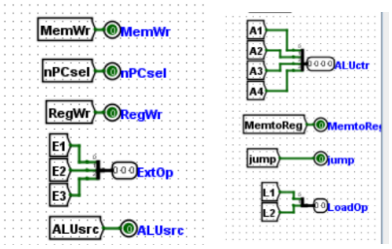





opcode
E1
opcode
opcode
E2
opcode
opcode
E3
opcode
nPCsel


opcode
jump
opcode
funct3
funct3
L1
opcode
funct3
L2
opcode

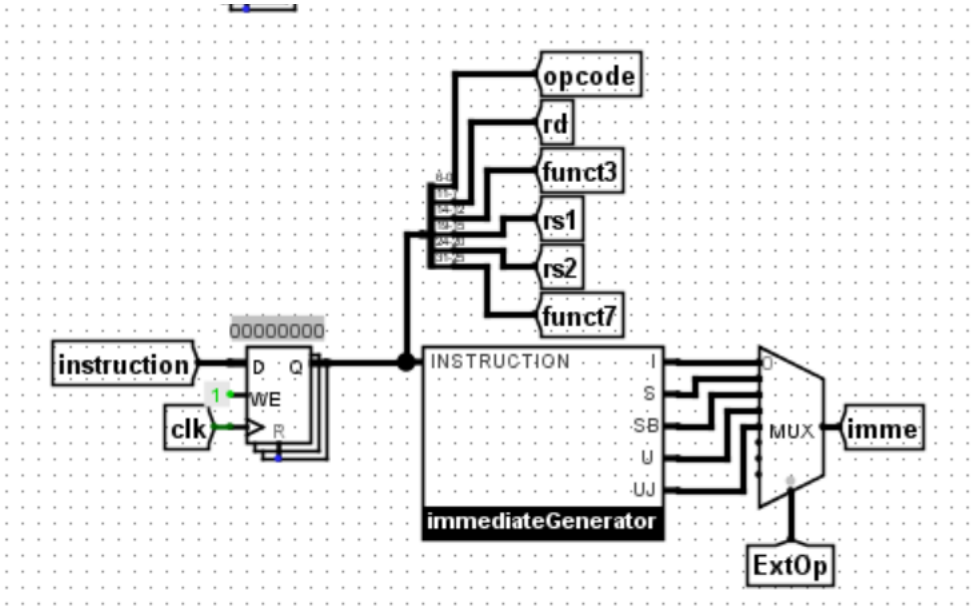


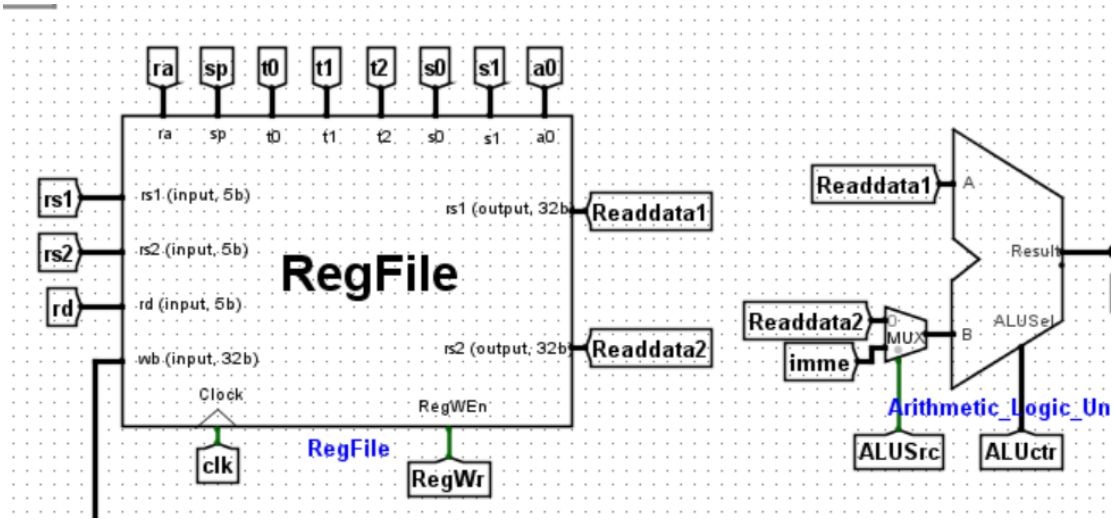

MemWr
MemWr
nPCsel
nPCsel
RegWr
RegWr
E1
E2
E3
ExtOp
ALUsrc
ALUsrc
A1
A2
A3
A4
ALUctr
MemtoReg
jump
jump
L1
L2
LoadOp

4.3 数据通道

4.3.1 设计思路及内容

1）将 32 位指令分解成 opcode，rs1，rs2，funct3，funct7，rd。rs1，rs2，rd 连接到寄存器文件（RegFile）的输入端，寄存器文件输出端为 Readdata1 和 Readdata2。Readdata1 连接到 ALU 的第一个输入端，第二个输入端需要根据指令操作中是否涉及立即数的运算来判断，Readdata2 和之前经过符号拓展的 32 为立即数经过一个二路复用器来选择哪个作为 ALU 的第二个输入，判断依据为 ALUsrc。如下图：

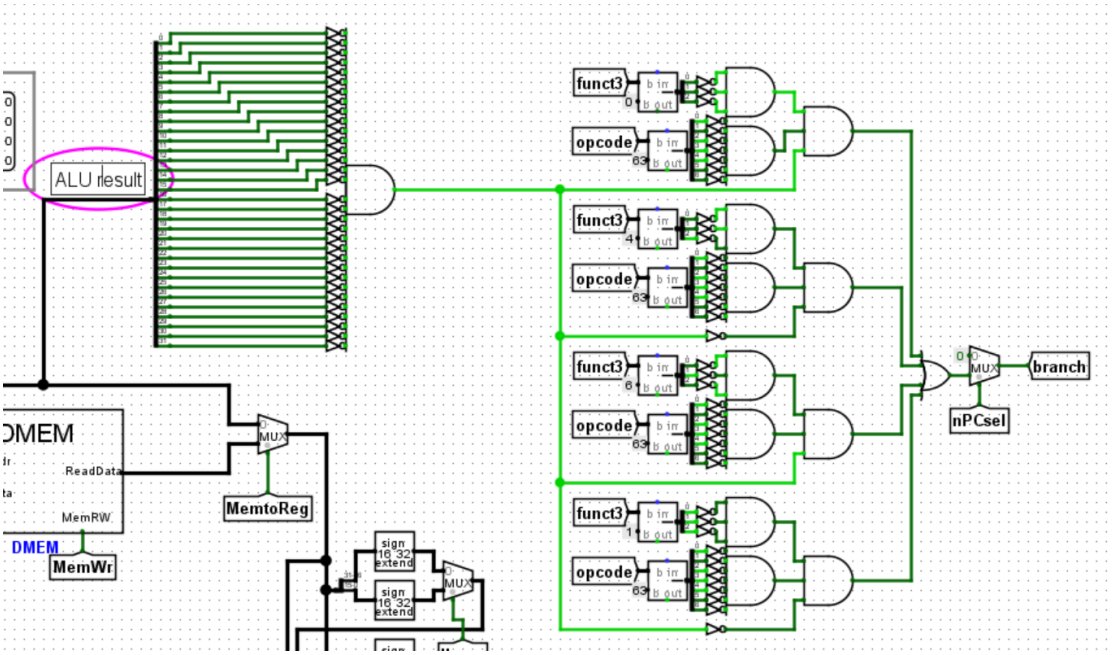




2）连接 ALUctr 控制信号选择 ALU 中要进行的运算。ALU 的输出结果可作为需要写入寄存器的数据（大多数指令），要写入数据的内存位置（store 指令），要读取数据的内存位置（load 指令），以及判断是否分支的依据（bne，beq，blt，bltu 等跳转指令）。

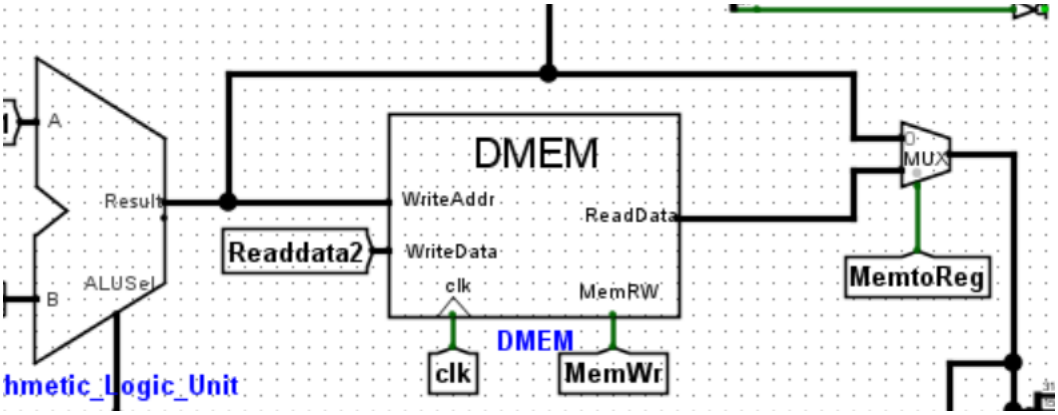
作为分支依据时，需要将 ALUresult 与 0 做比较，并根据当前指令内容来决定是否分支。对于 beq，ALUresult 要为 0 才会分支，而对于 bnq，ALUresult 不为 0 才会分支。对于 blt，

ALUresult 为 1 才会分支，对于 bltu，ALUresult 为 0 才会分支，如下图，分支标志为 **branch**。



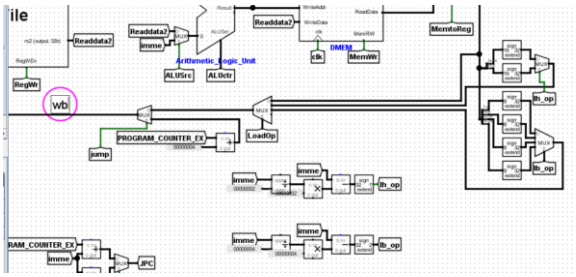
作为内存地址时，需要将输出连接到内存的 WriteAddr 上，将 Readdata2 连接到内存的 Writedata（store 指令将 Readdata2 存入到内存中），MemWr 作为是否写入内存的控制信号也要连接到内存上。

ALU 输出的结果和经过内存后的 ReadData 需要经过一个二路复用器并经过一系列操作后连接到寄存器文件的 wb 输入端，控制信号为 MemtoReg（是否由内存写入到寄存器）。



3）经过二路复用器输出的需要再进行一系列操作才能连接到 RegFile 的 wb 输入端。

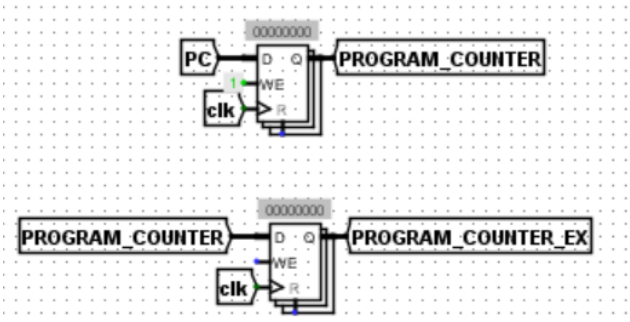
首先，结合 lh_op,lb_op 和多路复用器选择要装入的字节。其中 lh_op 为 imme 除以 2 的余数，lb_op 为 imme 除以 4 的余数。Lw，lh，lb 三个结果再通过一个四路复用器和 LoadOp 选择出合适的 wb，而 jal 和 jalr 指令存入 rd 的内容是 PC+4，所有还需再用一个二路复用器结合 Jump 控制信号决定最终的 wb，如下图。



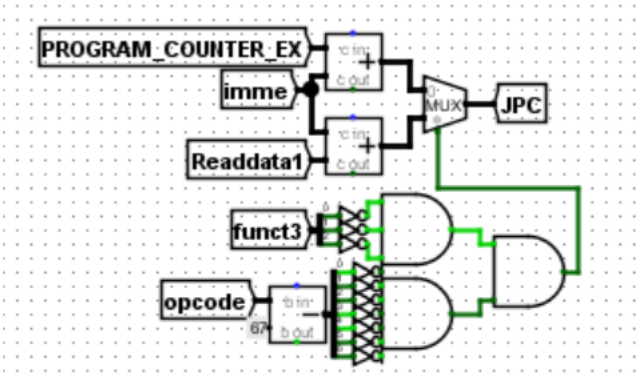
4）PC 部分的内容

为实现不同指令对 PC 的要求，将原有的 PC 部分改成如下：

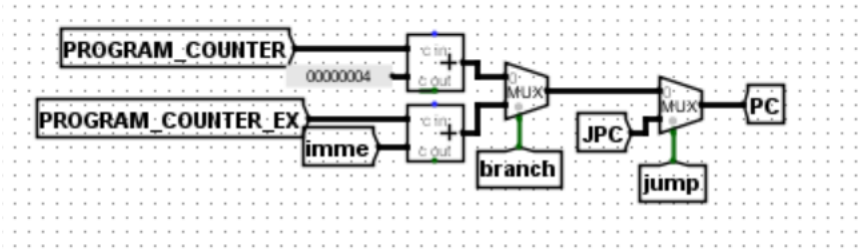
PROGRAM_COUNTER_EX 表示上一步的 PC 地址。当要分支或者跳转是就需要使用 PROGRAM_COUNTER_EX。



JPC 表示 Jump 指令跳转到的地址，由 jal 和 jalr 两条指令的内容可以得出，如下：

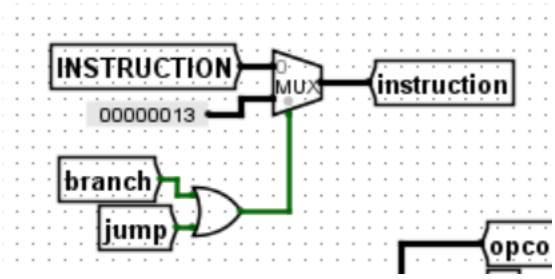


Branch 和二路复用器结合使用选择分支指令后的下一个地址，如下：



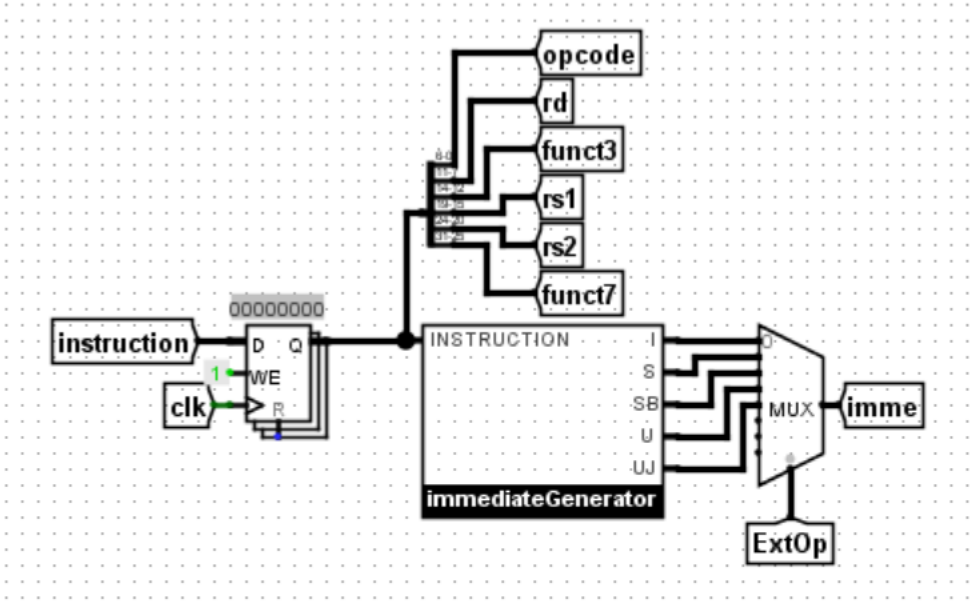
5）指令部分的内容

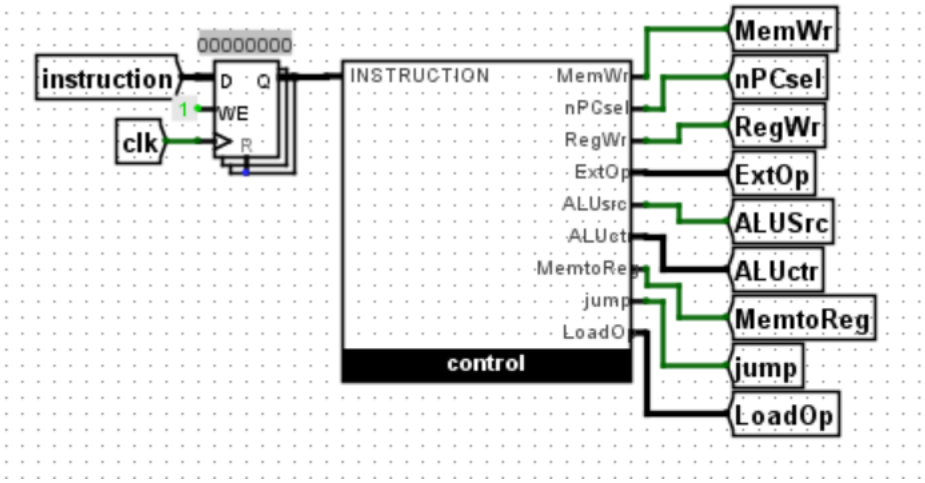
紧接在分支或跳转指令后面的指令，当真的需要分支或跳转时，该指令不能被执行，而流水线中必然会读取这条指令，所以需要分支或跳转时需要插入一个 nop，“kill”掉正在取的指令，具体实现如下。（0x00000013 是一条 nop 指令：addi x0, x0, 0 ）



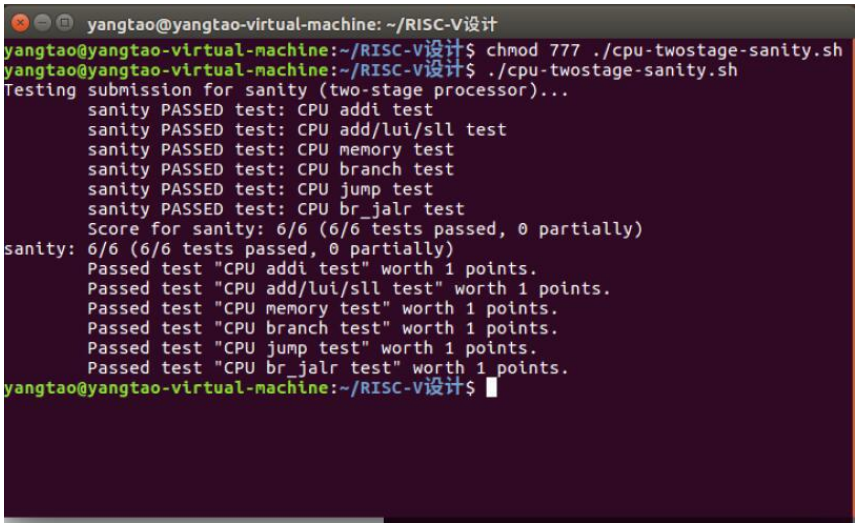
然后通过 instruction 得到各控制信号以及 rd，opcode，rs1，rs2，funct3 等内容。

如下图：


opcode
rd
funct3
rs1
rs2
funct7
00000000
instruction
D Q
WE
clk
R
INSTRUCTION
I
S
SB
U
UJ
immediateGenerator
MUX
imme
ExtOp


00000000
instruction
D Q
WE
clk
R
INSTRUCTION
MemWr
nPCsel
RegWr
ExtOp
ALUsrc
ALUct
MemtoRe
jump
LoadO
control
MemWr
nPCsel
RegWr
ExtOp
ALUSrc
ALUctr
MemtoReg
jump
LoadOp

4.3.2 测试结果

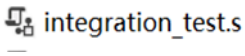

yangtao@yangtao-virtual-machine: ~/RISC-V设计
yangtao@yangtao-virtual-machine:~/RISC-V设计$ chmod 777 ./cpu-twostage-sanity.sh
yangtao@yangtao-virtual-machine:~/RISC-V设计$ ./cpu-twostage-sanity.sh
Testing submission for sanity (two-stage processor)...
sanity PASSED test: CPU addi test
sanity PASSED test: CPU add/lui/sll test
sanity PASSED test: CPU memory test
sanity PASSED test: CPU branch test
sanity PASSED test: CPU jump test
sanity PASSED test: CPU br_jalr test
Score for sanity: 6/6 (6/6 tests passed, 0 partially)
sanity: 6/6 (6/6 tests passed, 0 partially)
Passed test "CPU addi test" worth 1 points.
Passed test "CPU add/lui/sll test" worth 1 points.
Passed test "CPU memory test" worth 1 points.
Passed test "CPU branch test" worth 1 points.
Passed test "CPU jump test" worth 1 points.
Passed test "CPU br_jalr test" worth 1 points.
yangtao@yangtao-virtual-machine:~/RISC-V设计$

5. 测试用例的编写

测试用例都在“RISC-V 设计”文件夹中。

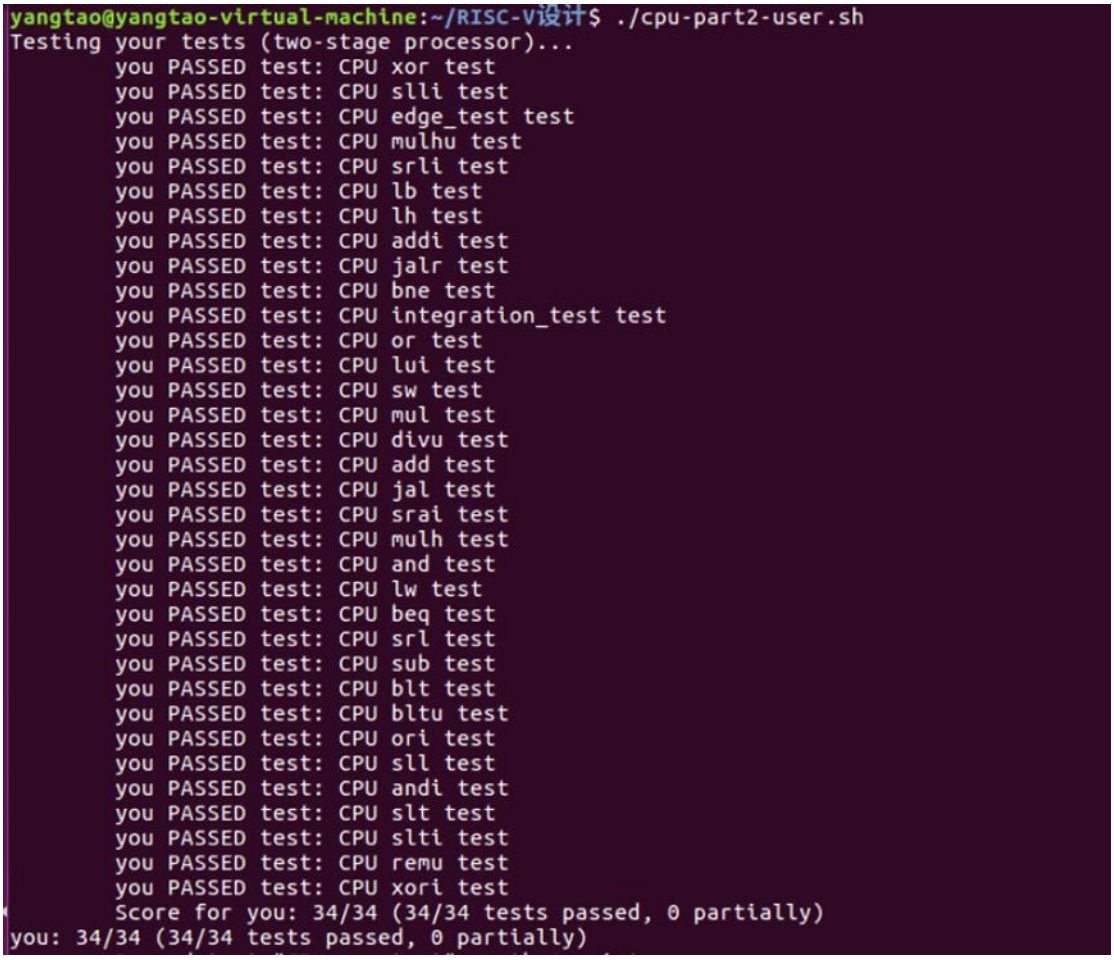
1. 单元测试

2. 集成测试



3. 边界测试

4. 自测结果



遇到的困难

在设计过程中，刚开始做数据通道和流水线的时候，对数据通道的具体内容仍不太清楚，漏掉了一些控制信号和过程，导致测试通不过。后通过参考课本上的内容以及与同学交流慢慢才一步一步完善了自己的设计。

未解决的问题

在编写测试用例的过程中，边界测试只测试了 sw，测试其他指令时总会报错，不明白要怎么设计。目前问题尚未解决。提交的边界测试用例文件中只是简单的测试了一下 sw 指令。

设计体会

通过这次课程设计，更加清楚的明白了 CPU 流水线中运行的流程，对这学期学的内容有了理解了更多，之前一直不怎么会使用寄存器，通过这次课程设计也学习和了解到了一些使用寄存器的方法和场景，了解了“加”，“减”，“乘”，“除”，位拓展器，移位（shifter）等子电路的使用方法，学会了使用这些子电路。

在这次计算机组成原理的课程设计的实践过程中，遇到了许多大大小小的问题，基本都是对课本内容不熟悉造成的。通过这次课程设计，也明白了清楚了解课本知识的重要性，要充分利用现有的资源来学习。