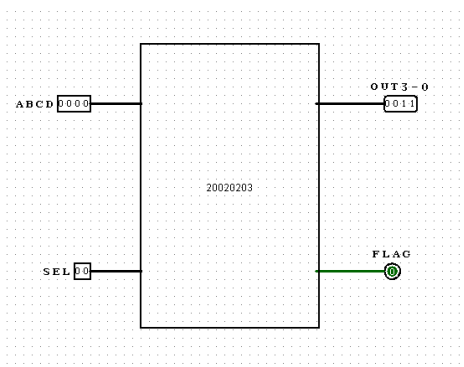


ALU、译码器、编码器、立即数扩展的设计与综合验证

一、基于 Logisim 平台的电路测试

1、应用测试截图



2、电路测试填写下表

				当 SEL=00 时	当 SEL=01 时	当 SEL=10 时	当 SEL=11 时	
输入				输出	输出	输出	输出	输出
A	B	C	D	OUT3-0	OUT3-0	OUT3-0	OUT3-0	FLAG
0	0	0	0	0011	0000	0000	0000	0
0	0	0	1	0100	0001	0001	0000	0
0	0	1	0	0101	0010	0011	0000	0
0	0	1	1	0110	0011	0010	0000	0
0	1	0	0	0111	0100	0110	0000	0
0	1	0	1	1000	1011	0111	0000	0
0	1	1	0	1001	1100	0101	0000	0
0	1	1	1	1010	1101	0100	0000	0
1	0	0	0	1011	1110	1100	0000	0
1	0	0	1	1100	1111	1000	0000	0
1	0	1	0	0000	0000	0000	0000	1
1	0	1	1	0000	0000	0000	0000	1
1	1	0	0	0000	0000	0000	0000	1
1	1	0	1	0000	0000	0000	0000	1
1	1	1	0	0000	0000	0000	0000	1
1	1	1	1	0000	0000	0000	0000	1

3、分析逻辑功能

- (1)SEL=00 时,将输入的自然二进制码转换为余 3 码输出;并且转换功能生效时 FLAG 输出 0, 否则输出 1;
- (2)SEL=01 时,将输入的自然二进制码转换为 2421 码输出;并且转换功能生效时 FLAG 输出 0, 否则输出 1;
- (3)SEL=10 时,将输入的自然二进制码转换为格雷码(循环码)输出;并且转换功能生效时 FLAG 输出 0, 否则输出 1;
- (4)SEL=11 时,对输入的自然二进制码进行检测,当表示的十进制数为 0~9 时, FLAG 输出 0, 当为 10~15 时, 输出 1。

4、理解

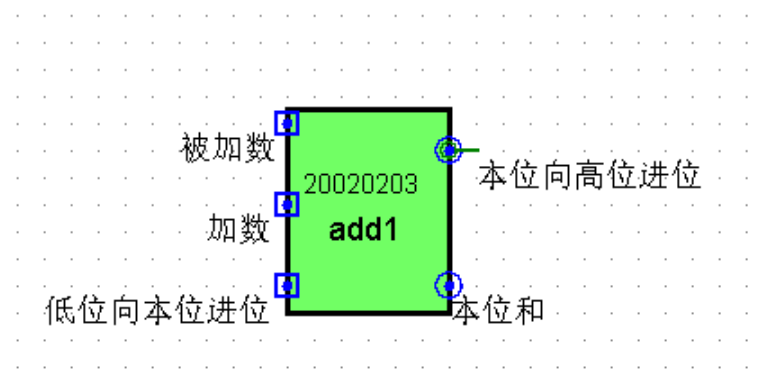
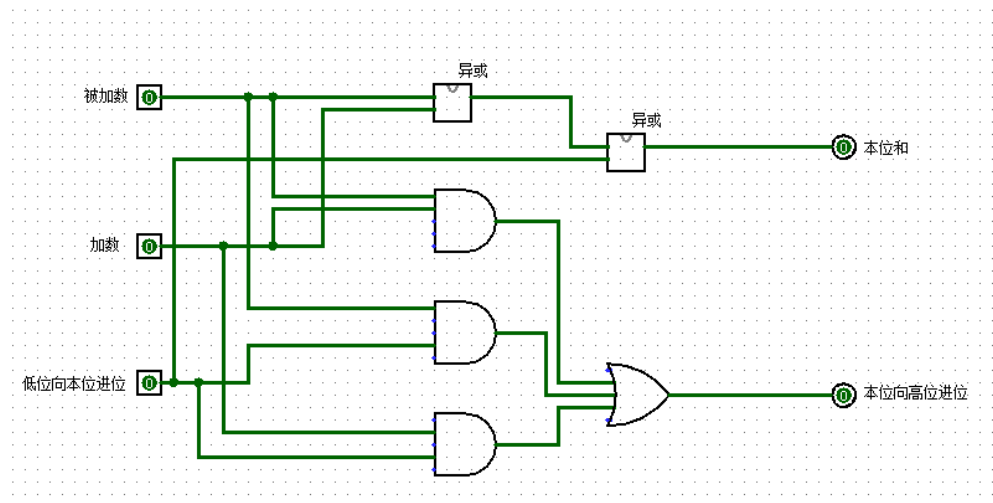
该电路可以实现多种类型的代码转换电路, 首先分别构建了各自的代码转换电路,

然后通过选择信号 SEL 进行控制输出，使用“数据选择器”模块，当 SEL 不同时，对应得到输出信号也就不同，实现了多种类型代码转换的功能。

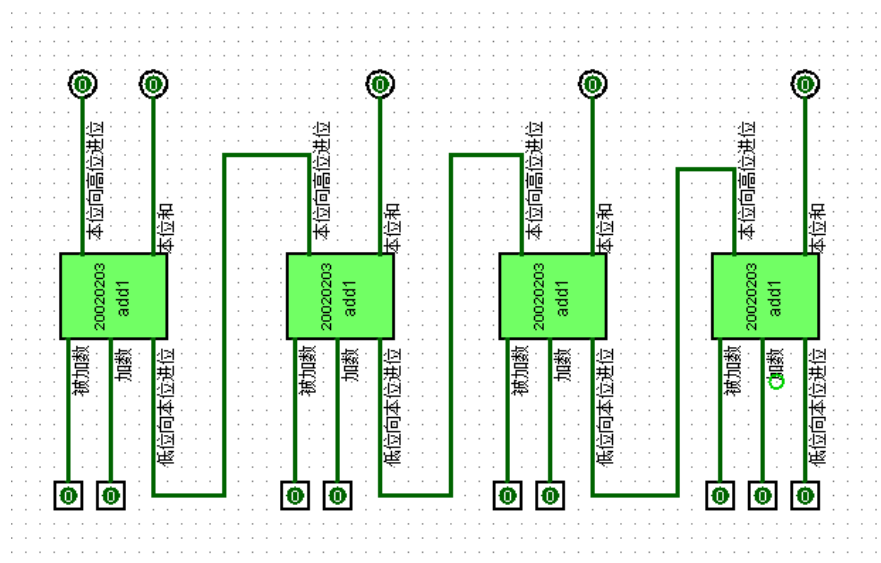
二、设计能完成四种运算的 32 位 ALU

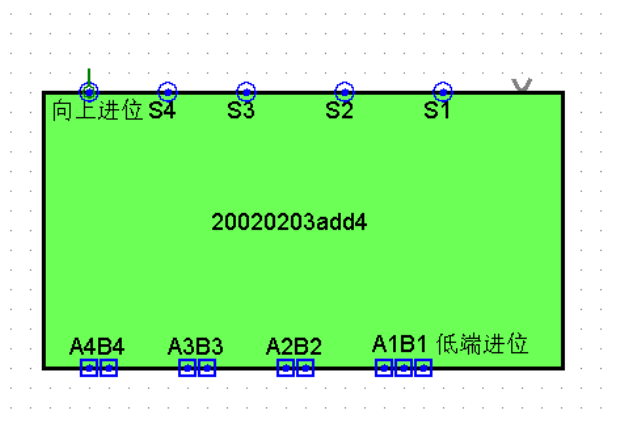
1、全加器

(1) 1 位全加器

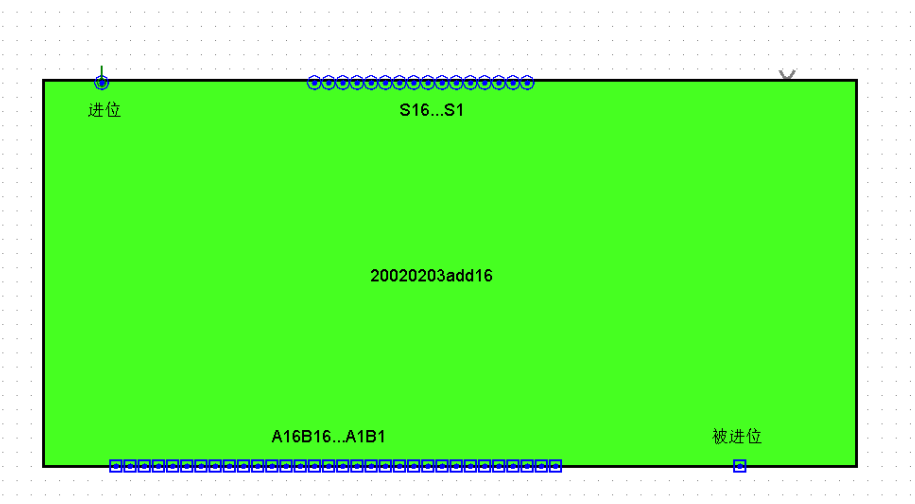
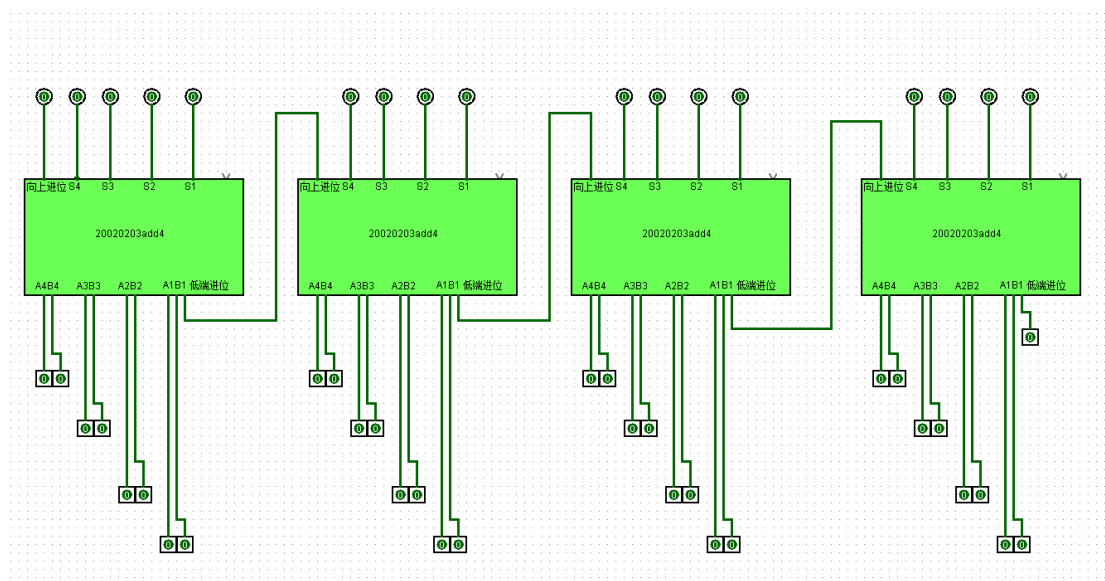


(2) 4 位全加器

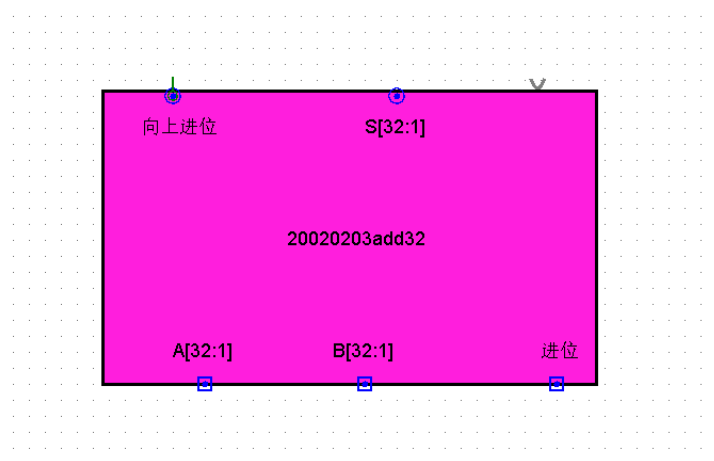
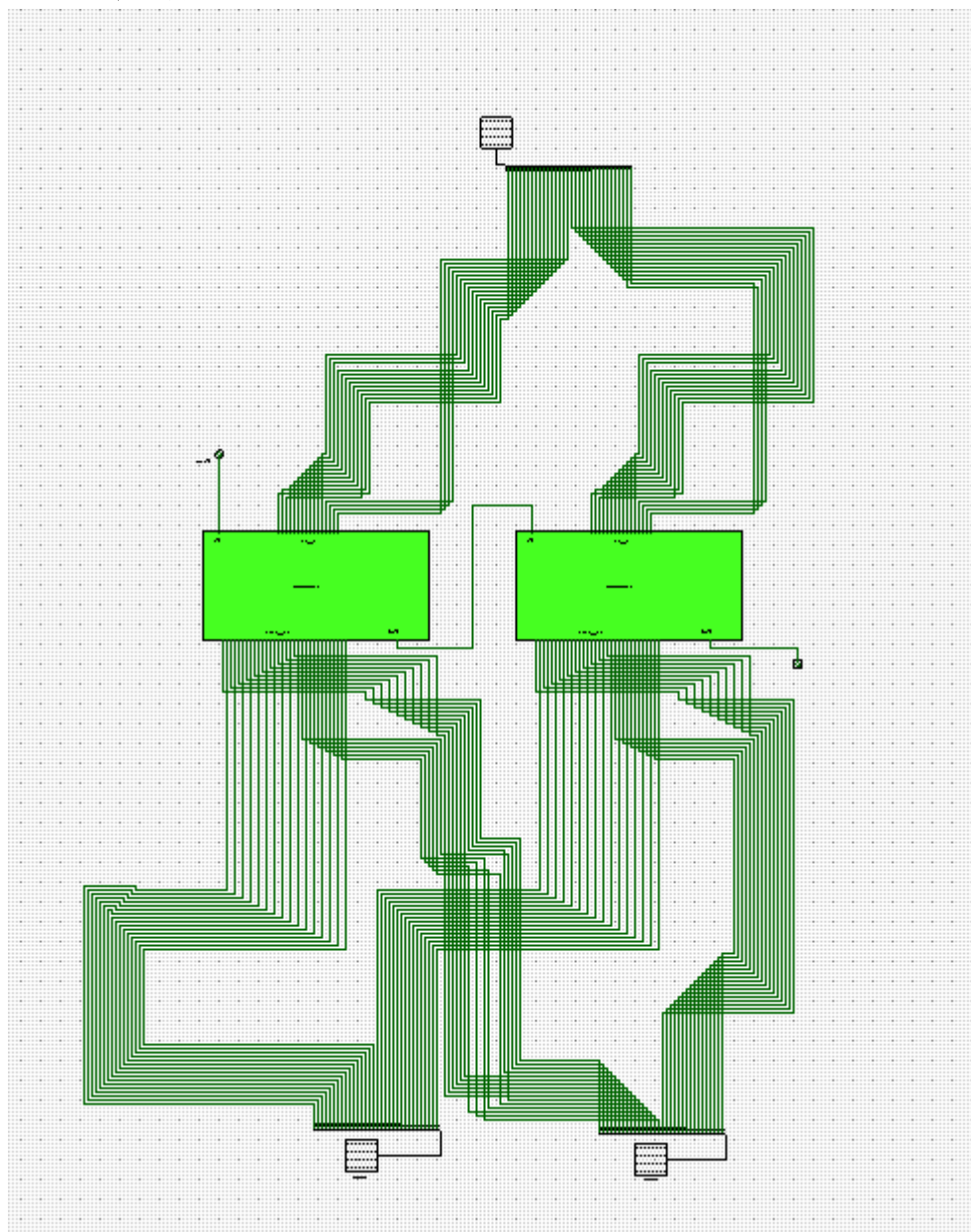




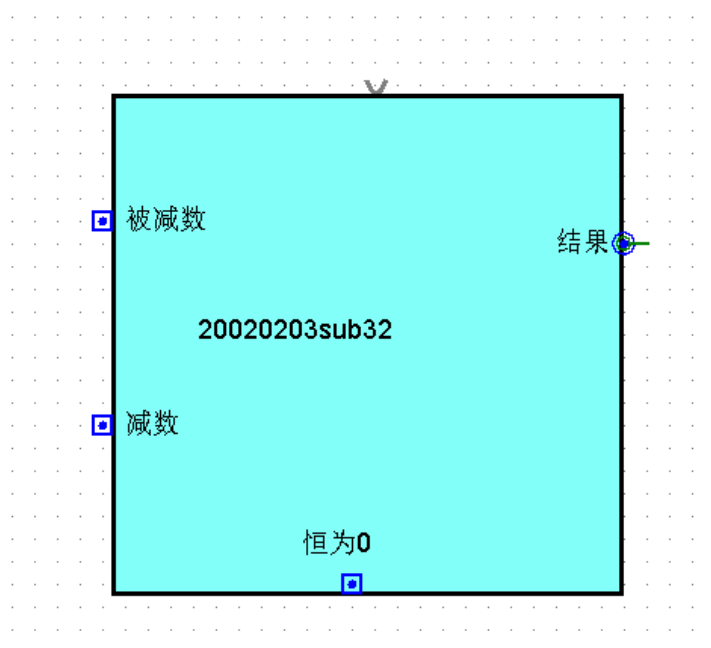
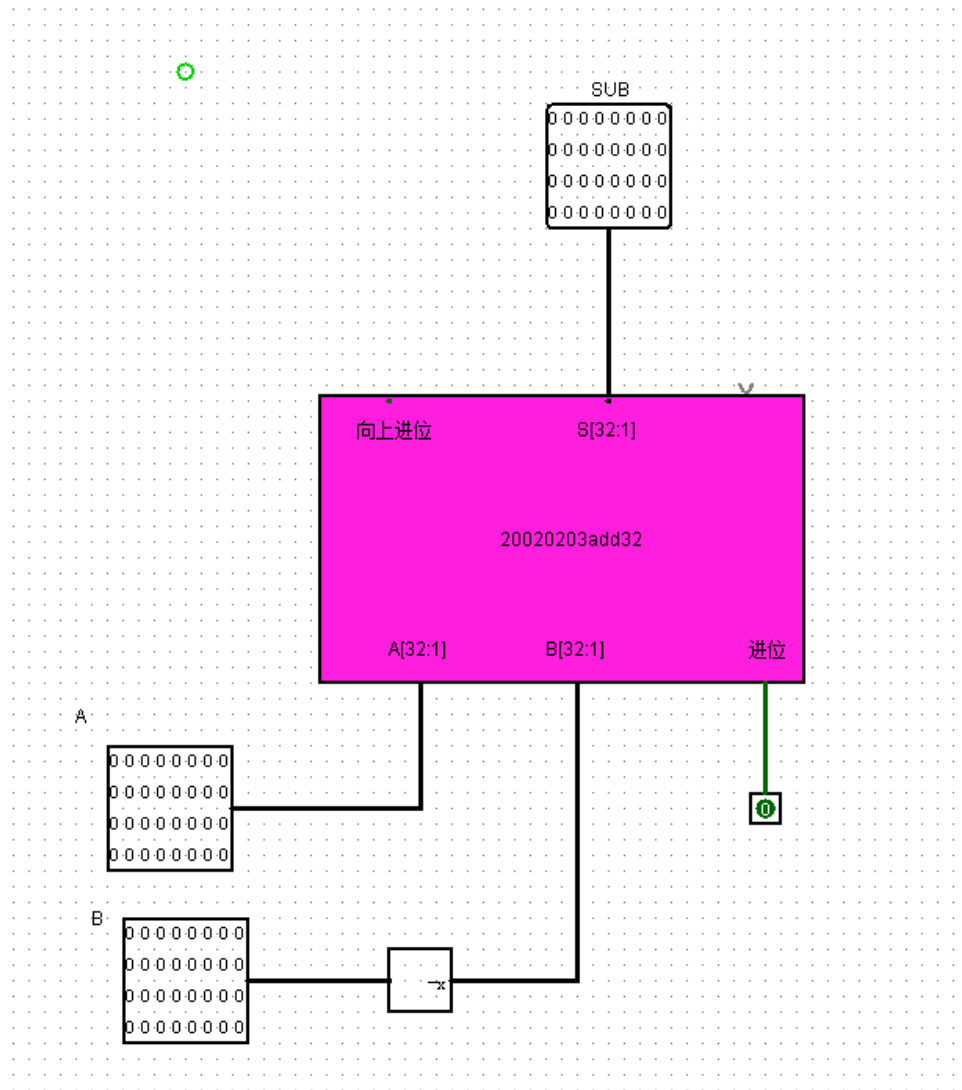
(3) 16 位全加器



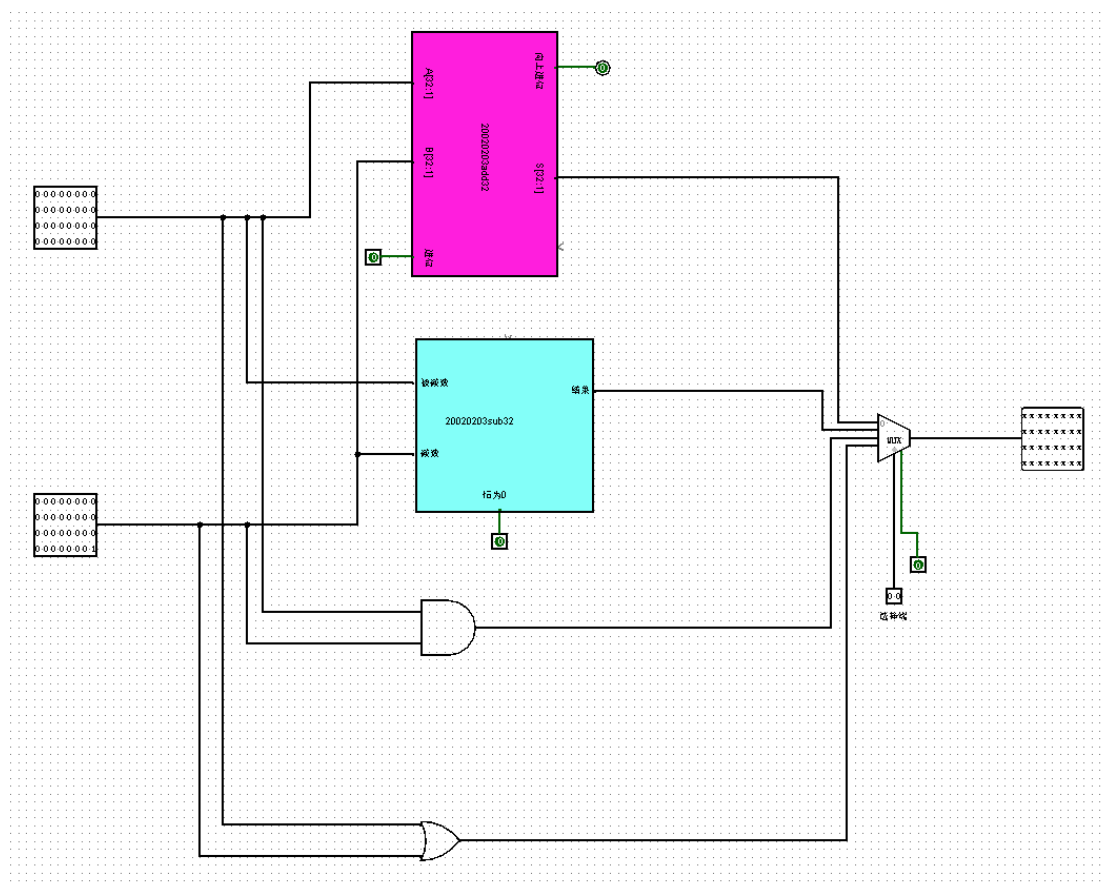
(4) 32 位全加器



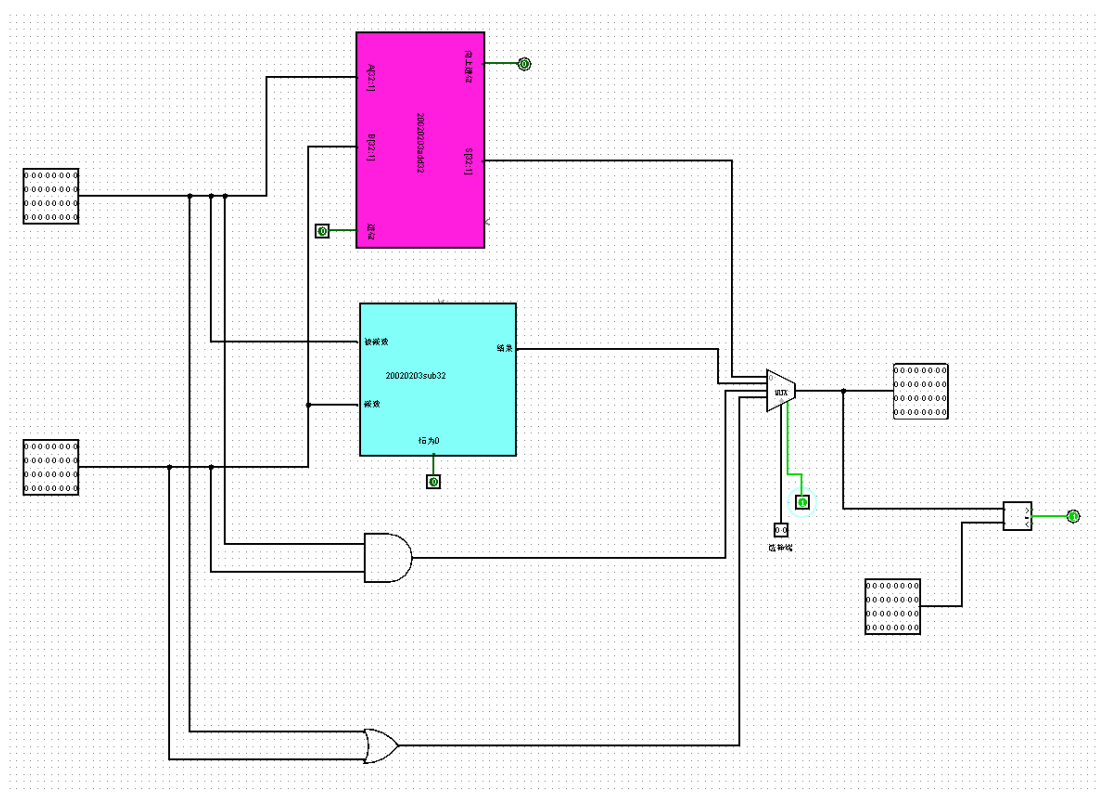
2、32 位全减器



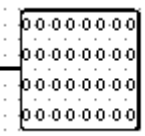

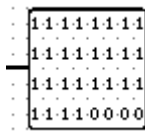

3、能完成四种运算的 ALU

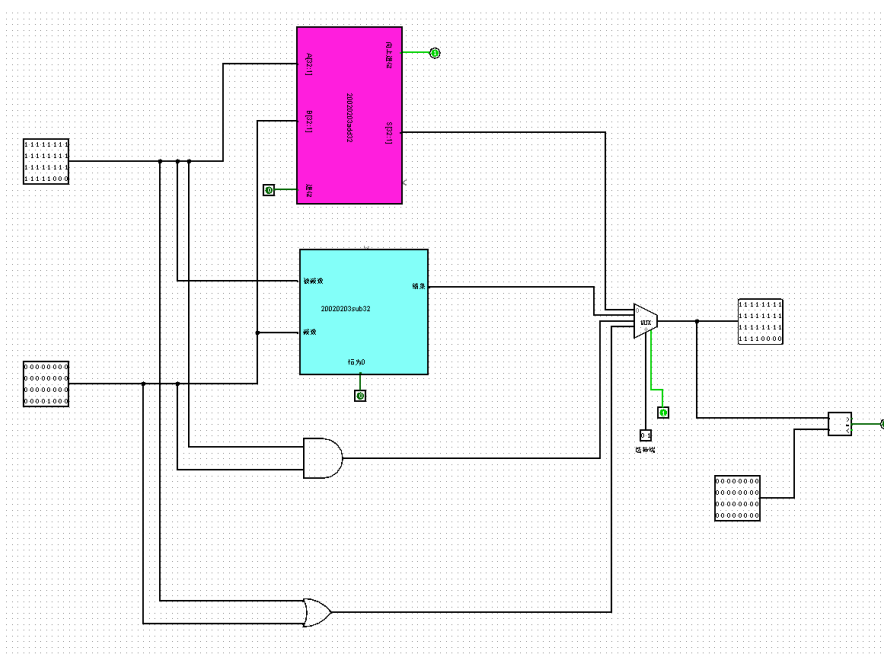
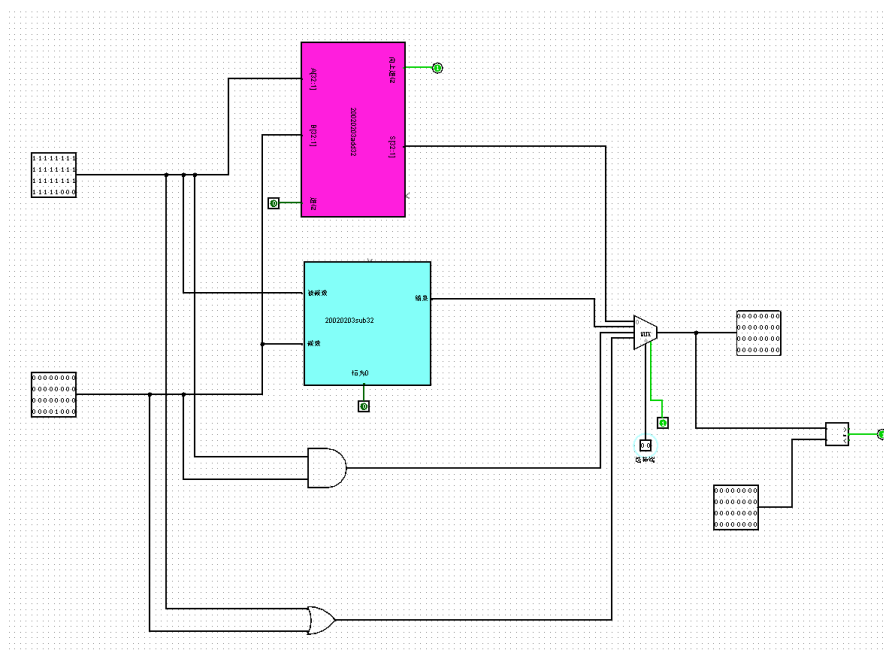


4、添加运算结果为零的标志信号 zero



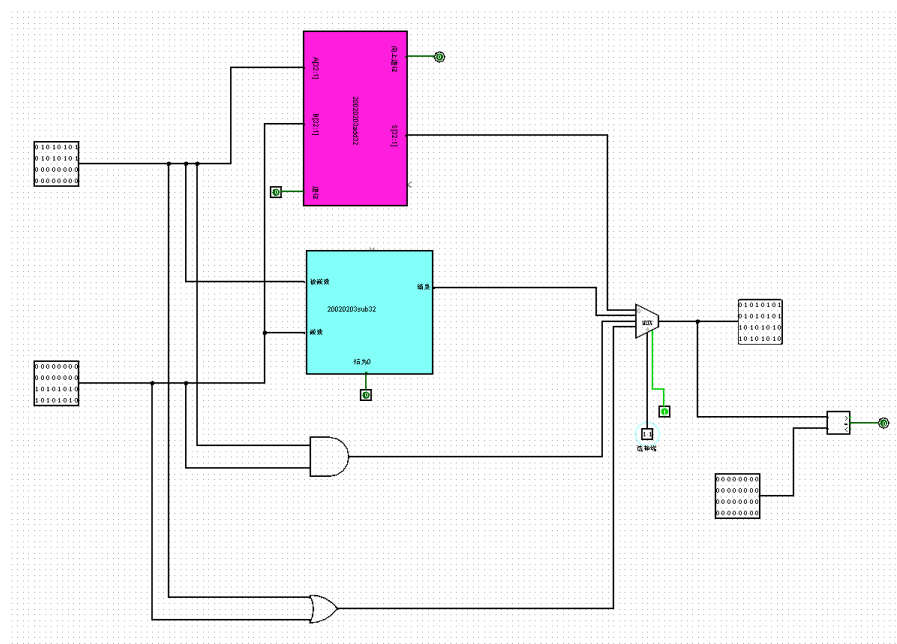
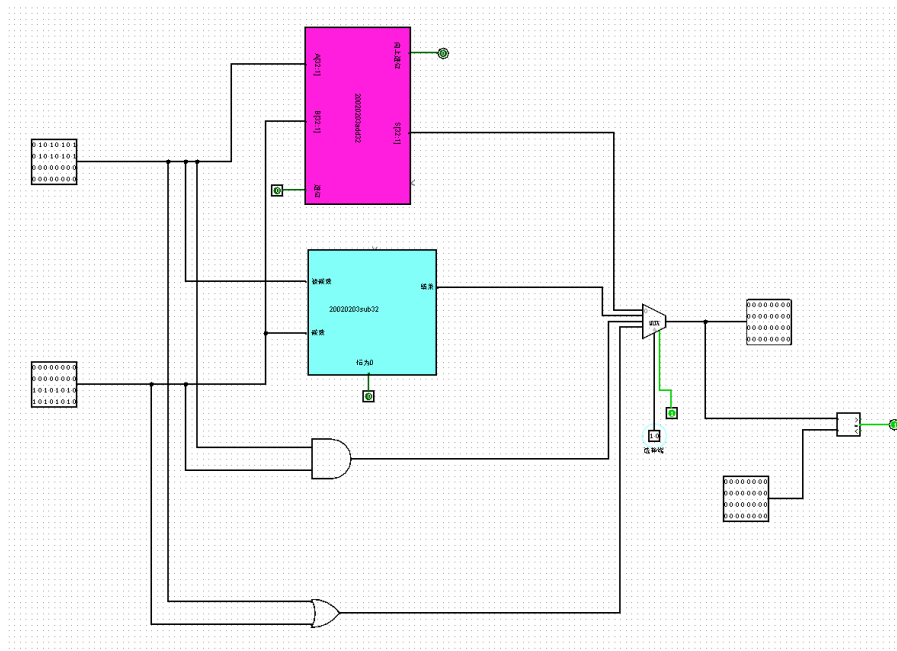
5. 算术运算（补码）测试：若 $(A)_{10} = -8$ ， $(B)_{10} = 8$ ，

ALU_sel	ALU_out	$(ALU_out)_{10}$	zero
00		0	
01		-16	



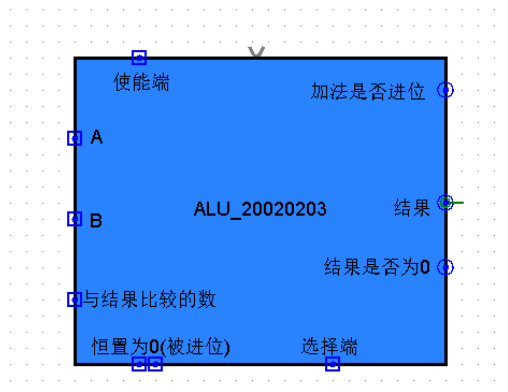
6. 逻辑运算测试：若(A)₁₆=55550000, (B)₁₆=0000AAAA,

ALU_sel	ALU_out
10	<pre> 00000000 00000000 00000000 00000000 </pre>
11	<pre> 01010101 01010101 10101010 10101010 </pre>

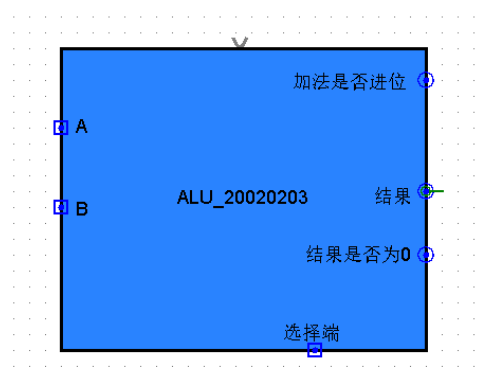


7. 编辑逻辑符号 (ALU-学号), 截图备用。

(1)更改前



(2)更改后(为部分地方赋予常量值, 这样就可以减少一部分输入端)

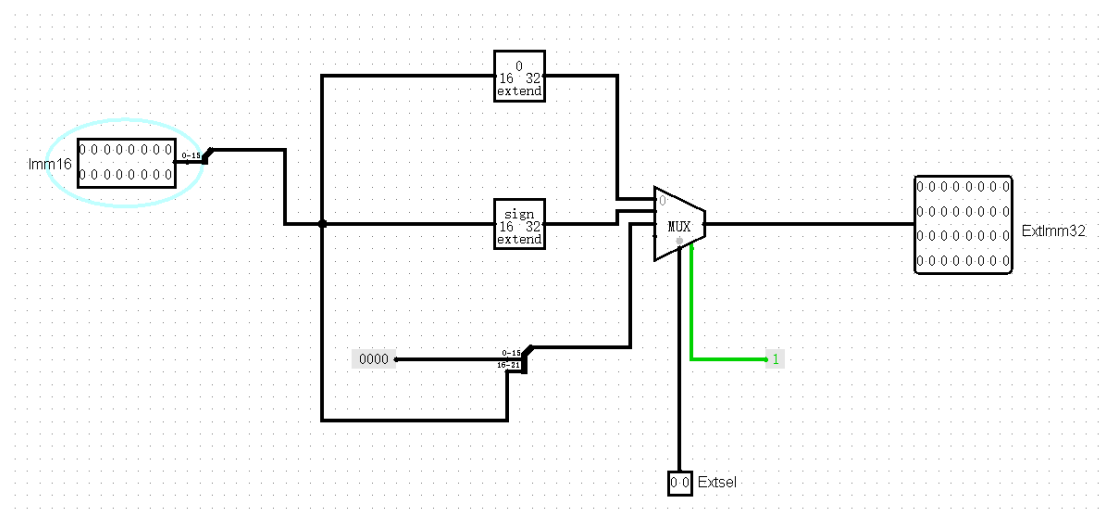


8. 将这个 ALU 的 Verilog HDL 描述补充完整 (模块名含学号)。

```
module ALU_20020203(A,B,ALU_sel,ALU_out,zero);
    input [32:1] A,B;
    input [1:0] ALU_sel;
    output [32:1] ALU_out;
    output zero;
    reg zero;
    assign zero=(ALU_out==32'b0)?1:0;
    always @(A or B or ALU_sel)
        case (ALU_sel)
            0:ALU_out=A+B;
            1: ALU_out=A-B;
            2: ALU_out=A&B;
            3: ALU_out=A|B;
            default: ALU_out=32'b0;
        endcase
endmodule
```

三、立即数扩展电路的设计与验证

1、利用 Logisim 中的“分线器”和“选择器”，构造立即数扩展电路。



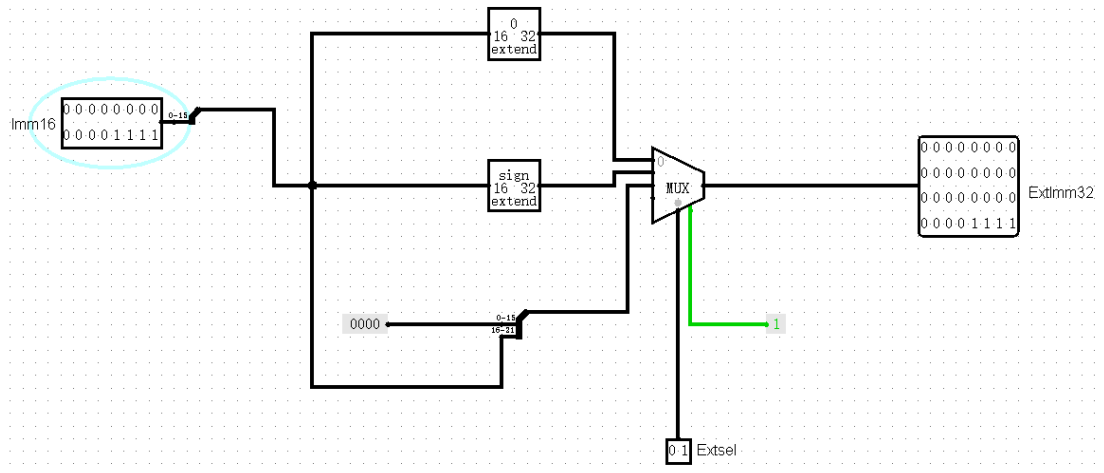
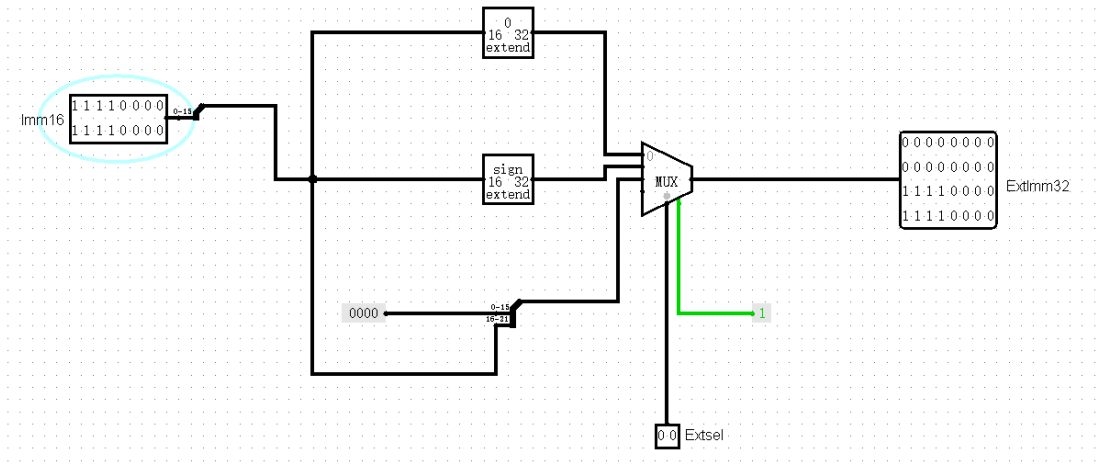
2、功能表

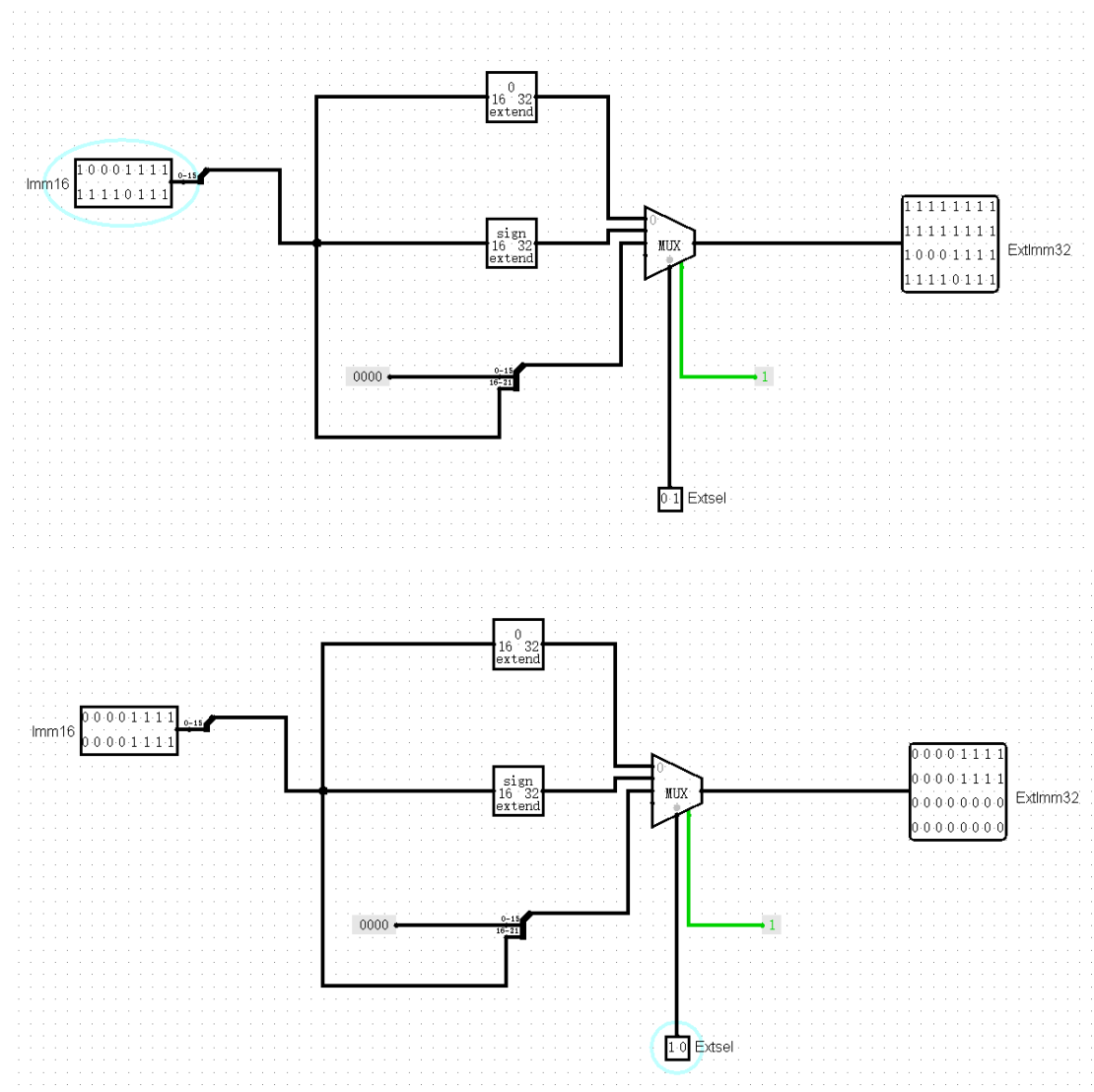
信号名	位宽	方向	说明
Imm16	16	输入	来自指令寄存器的 16 位立即数
Extsel	2	输入	00: 无符号扩展, 将 16 位立即数进行 0 扩展至 32 位立即数; 01: 符号扩展, 将 16 位补码立即数扩展成 32 位补码立即数; 10: 低位 0 扩展, 将 16 位立即数移至 32 位立即数的高 16 位, 低 16 位补 0。
ExtImm32	32	输出	扩展后的 32 位立即数

3、模拟验证, 填表下表

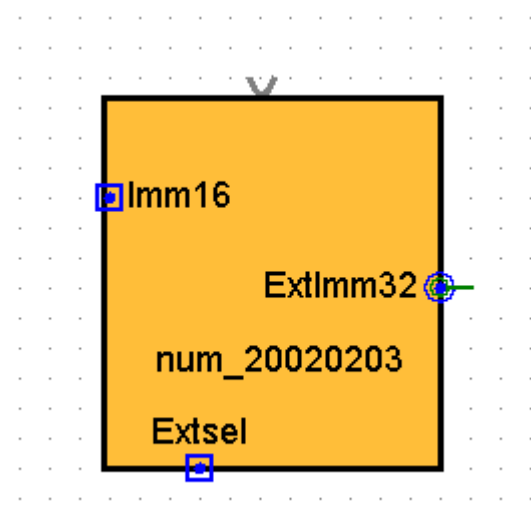
Imm16	Extsel	ExtImm32
1111000011110000	00	
0000000000001111	01	
1000111111110111	01	

0000111100001111	10	<div data-bbox="949 235 1181 369"> <pre> 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 </pre> </div> <div data-bbox="1093 302 1181 324">ExtImm32</div>
------------------	----	--





4、编辑逻辑符号，备用。

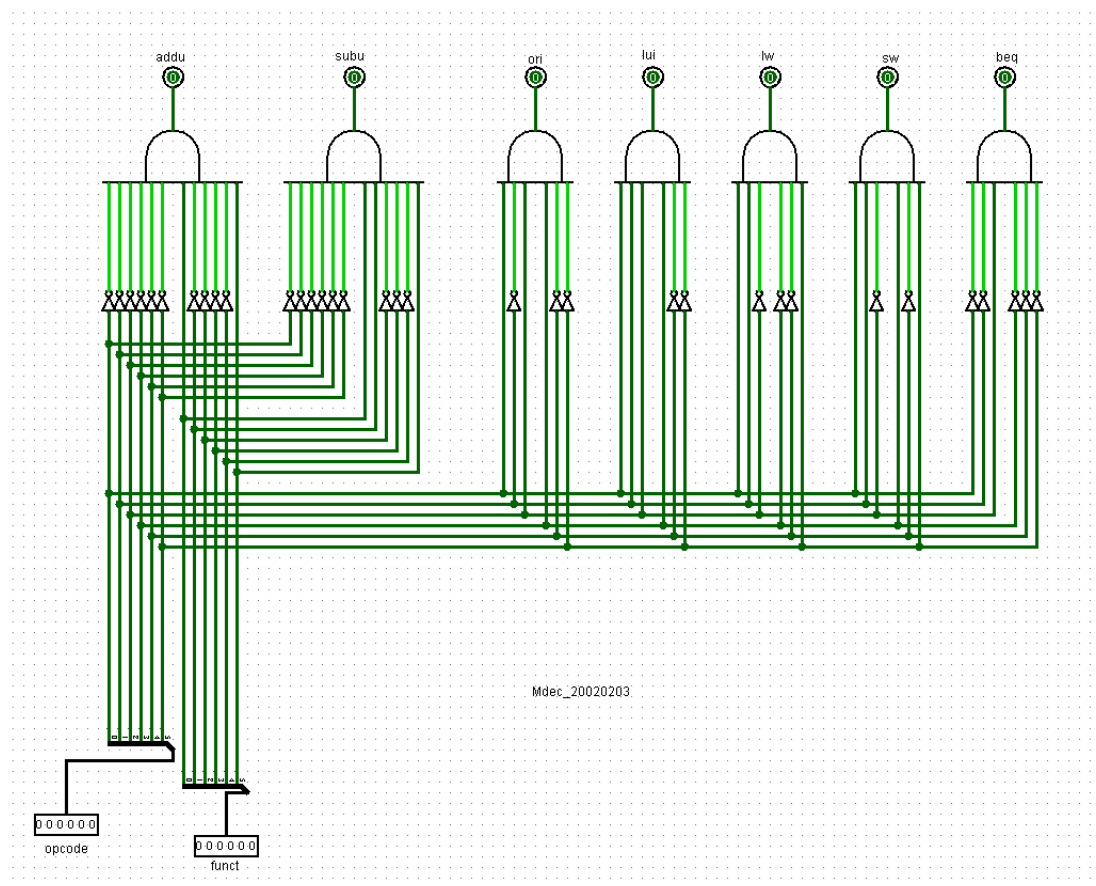


四、7 条 MIPS 指令的译码电路设计与验证

1. 7 条指令说明见相关 ppt，根据 7 条 MIPS 指令的特征码 “opcode” 和 “funct”，设计指令译码器。

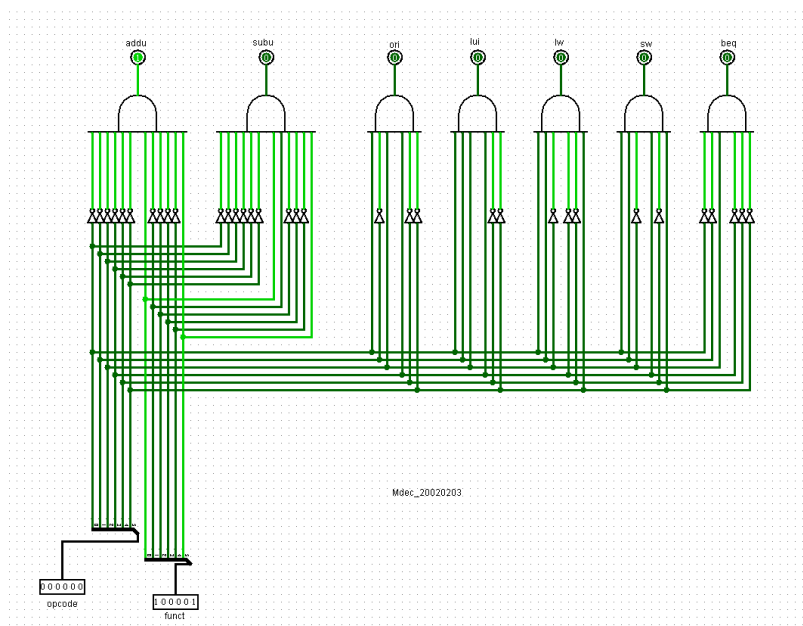
输入		输出						
		加法	减法	或立即数	立即数置高位	取字	存字	相等跳转
opcode	funct	addu	subu	ori	lui	lw	sw	beq
000000	100001	1	0	0	0	0	0	0
000000	100011	0	1	0	0	0	0	0
001101	X	0	0	1	0	0	0	0
001111	X	0	0	0	1	0	0	0
100011	X	0	0	0	0	1	0	0
101011	X	0	0	0	0	0	1	0
000100	X	0	0	0	0	0	0	1

【设计实现】

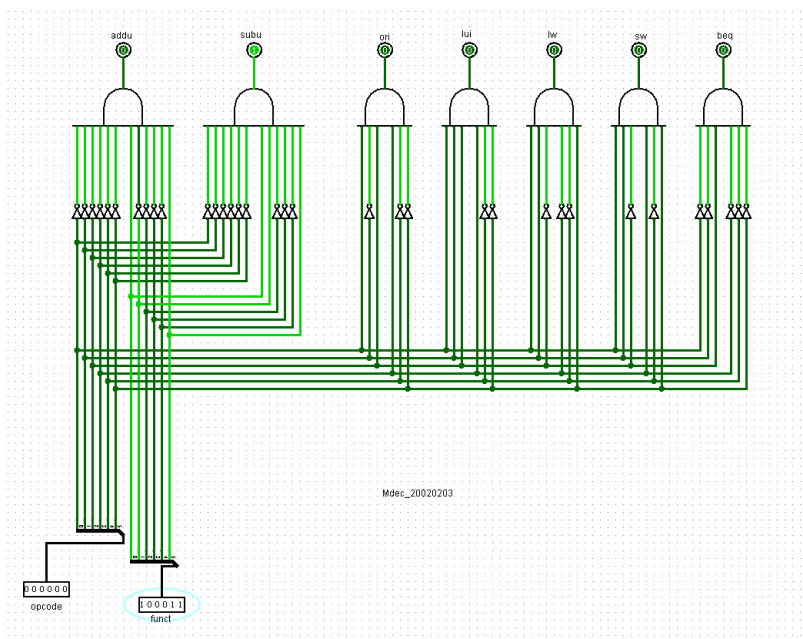


2. 在 Logisim 中设计实现指令译码器，进行验证，并截取正确结果电路图。

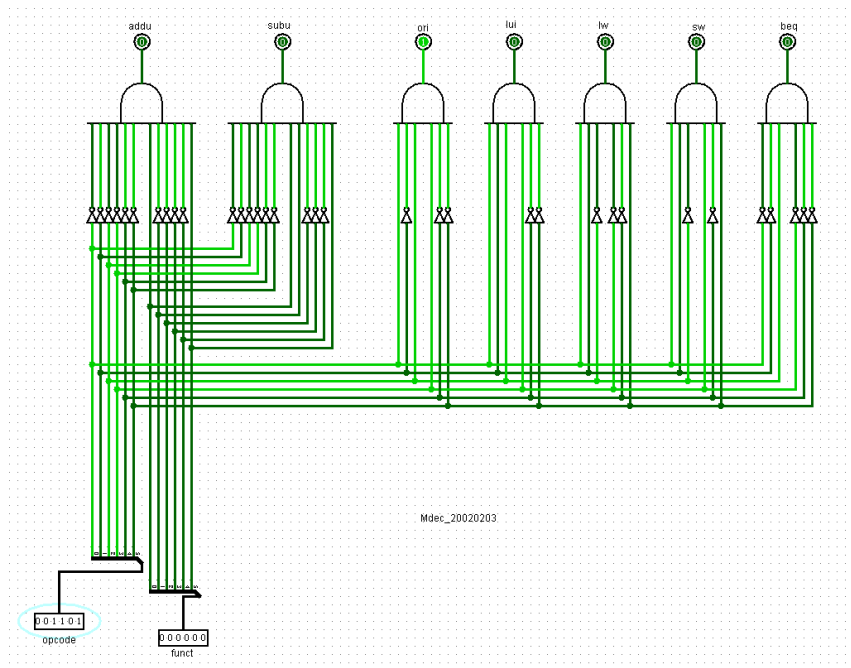
【addu】



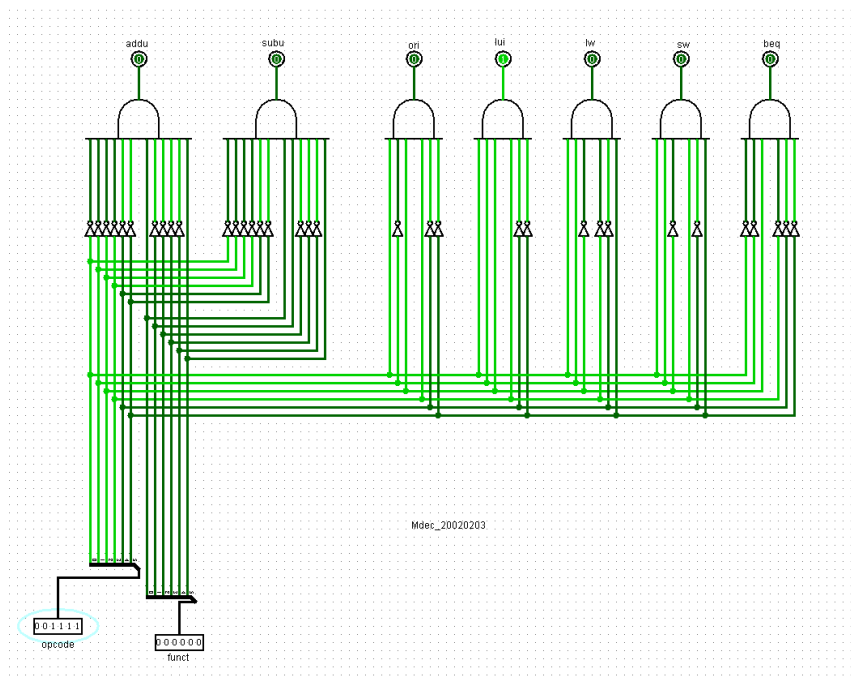
【subu】



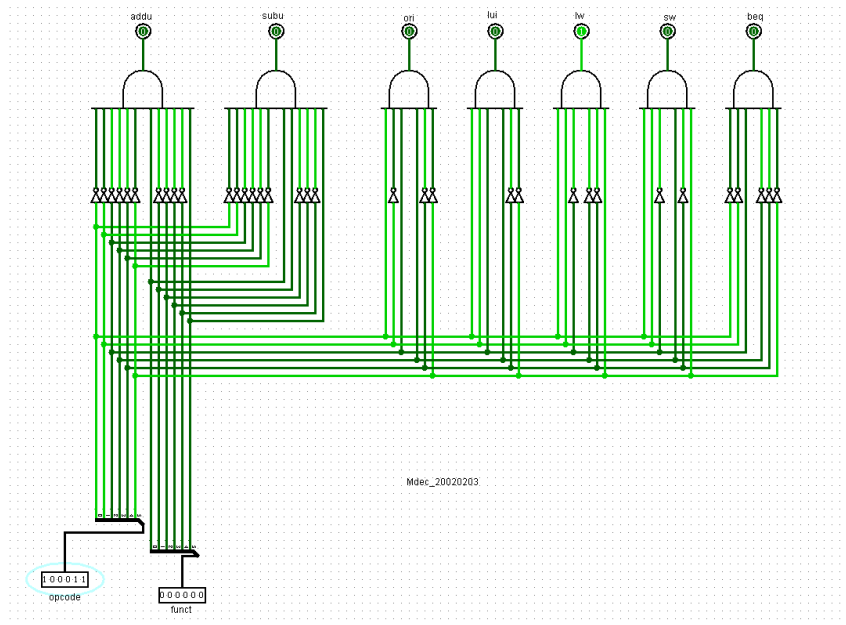
【ori】



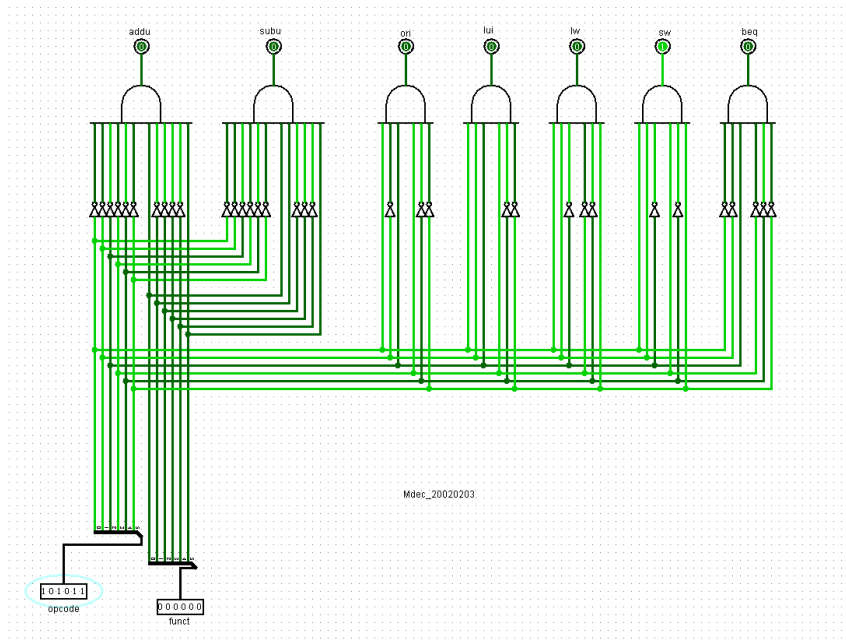
【lui】



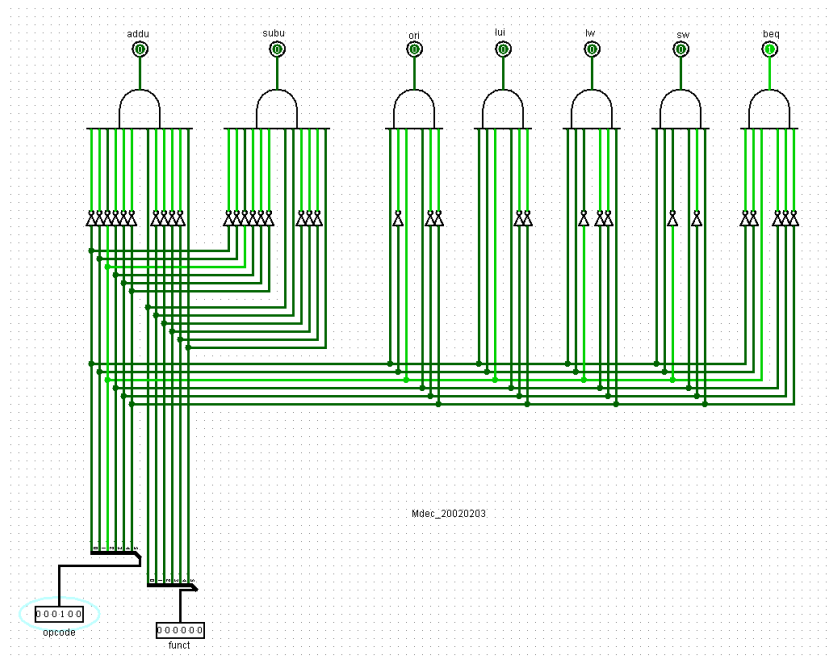
【lw】



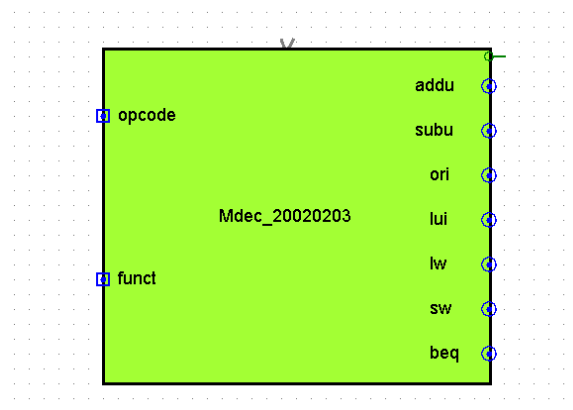
【sw】



【beq】



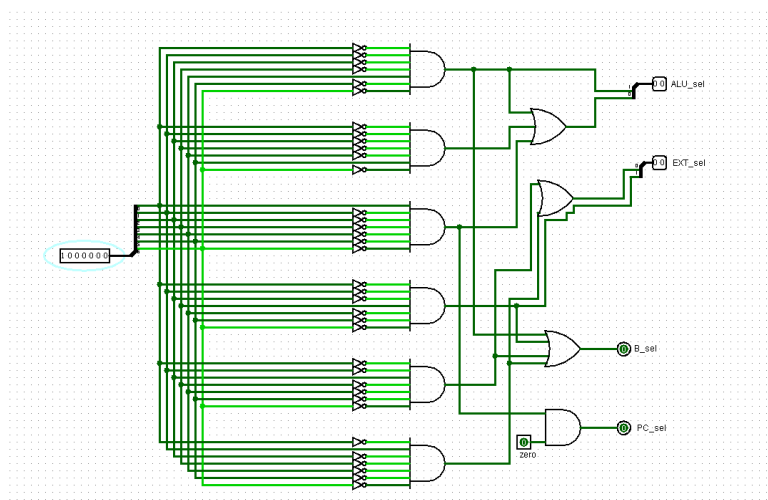
3. 编辑逻辑符号，备用。



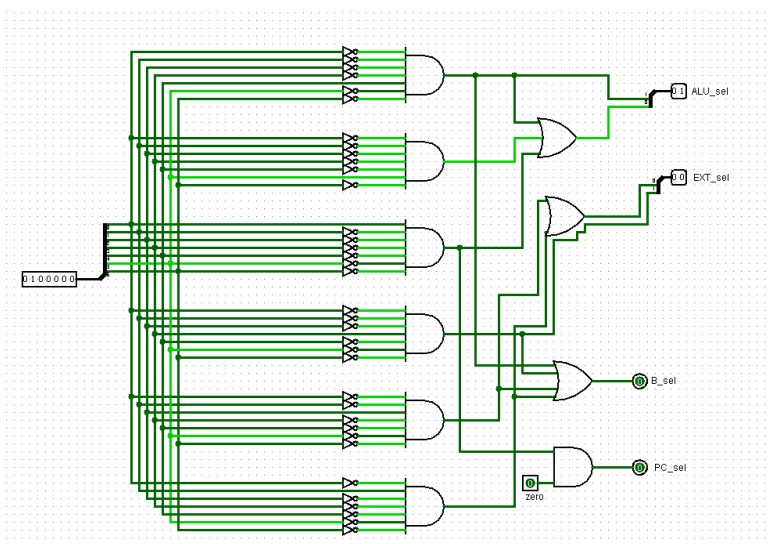
五、相关选择信号的编码器设计与验证

1、在 Logisim 中设计实现编码器，进行验证，并截取正确结果电路图。

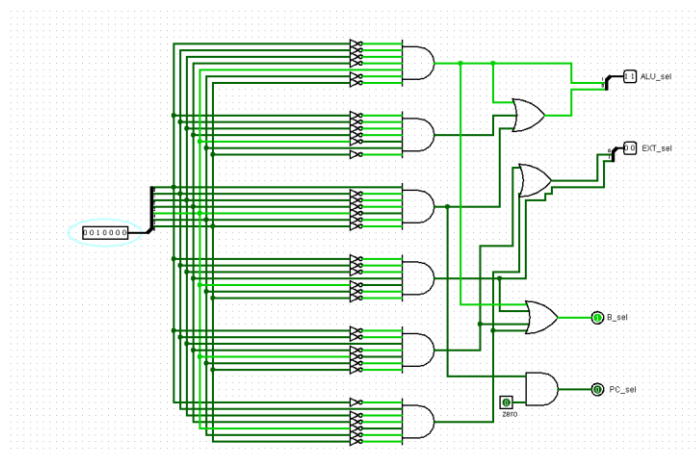
【addu】



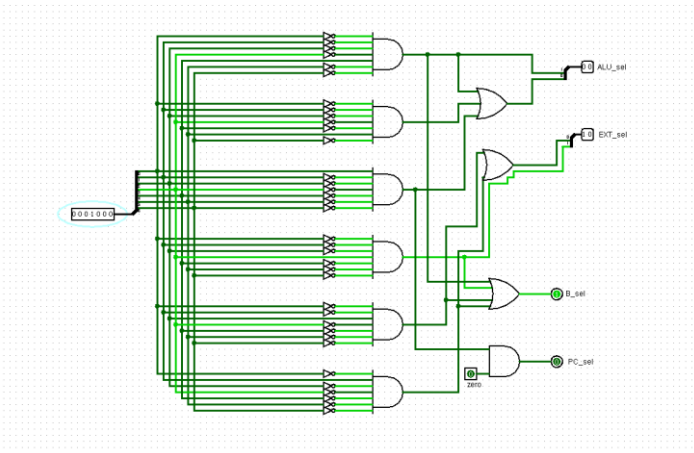
【subu】



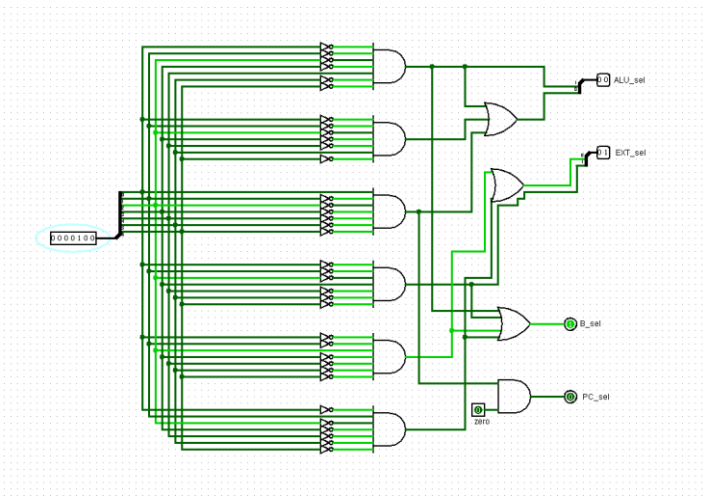
【ori】



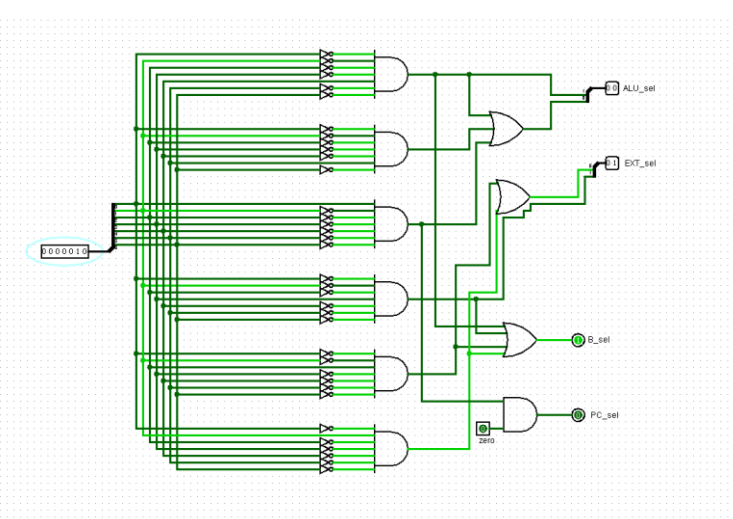
【lui】



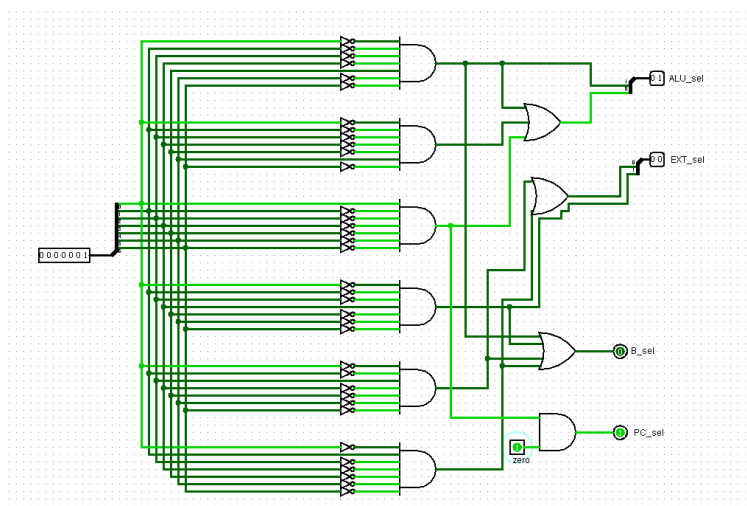
【lw】



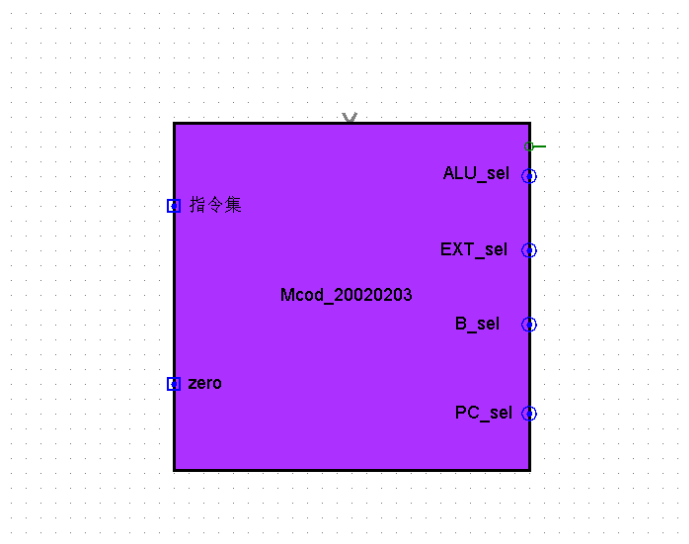
【sw】



【beq】

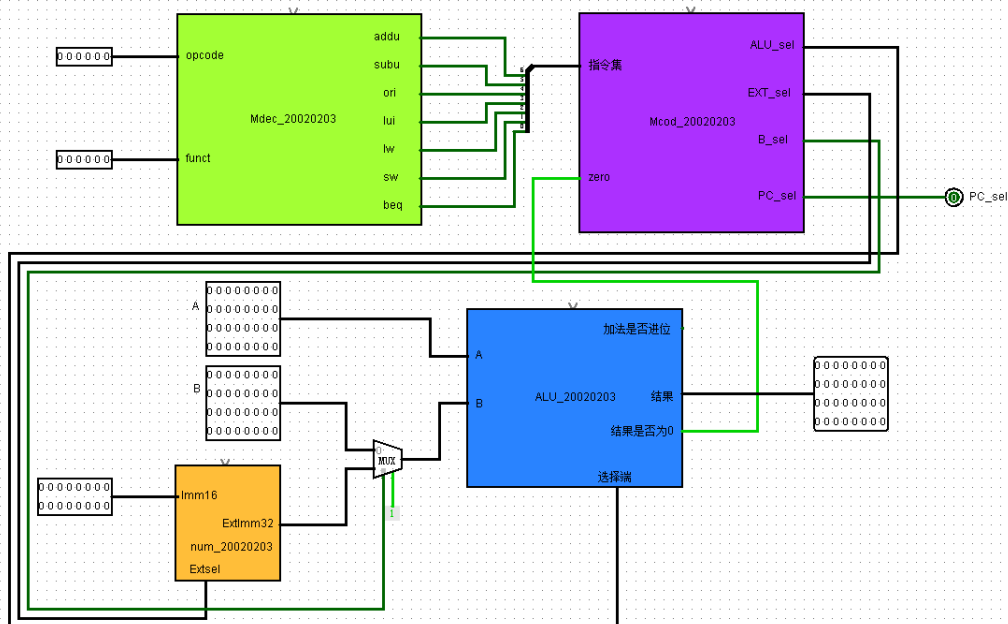


2、编辑逻辑符号（Mcod-学号），备用。

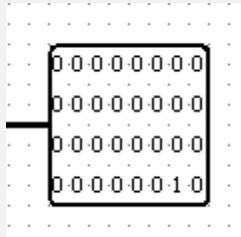
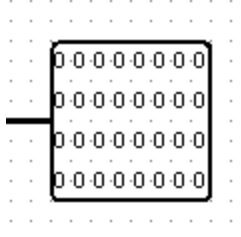
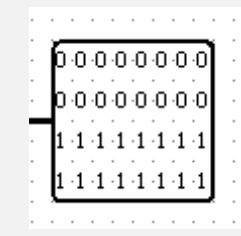


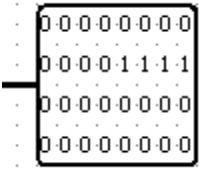
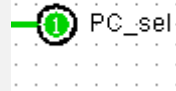
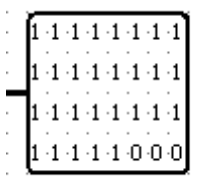
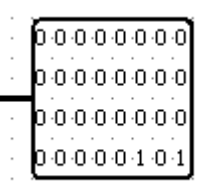
六、综合验证

1. 在 Logisim 中，按照图 1 进行电路（逻辑符号）连接。

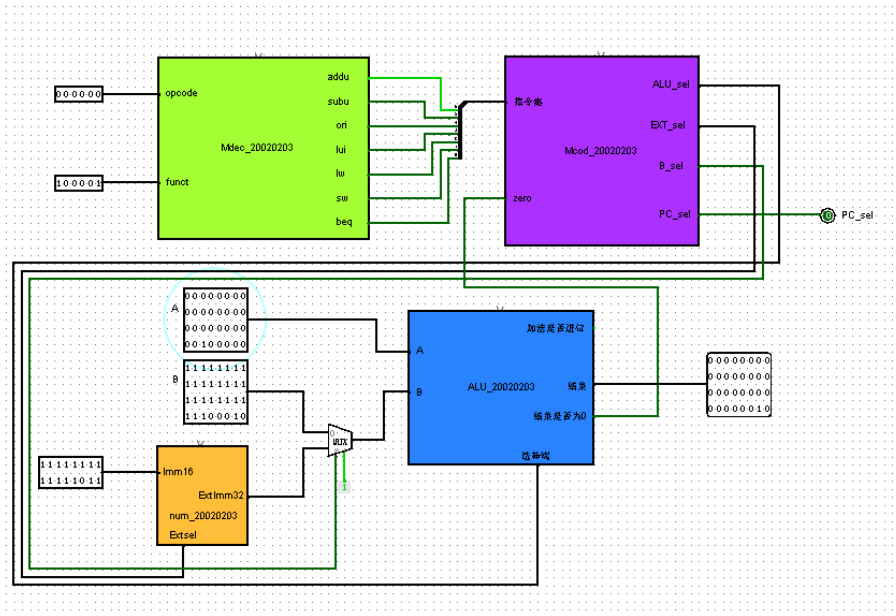


2. 按给定数据（注：十进制）进行测试并截取正确结果电路图。

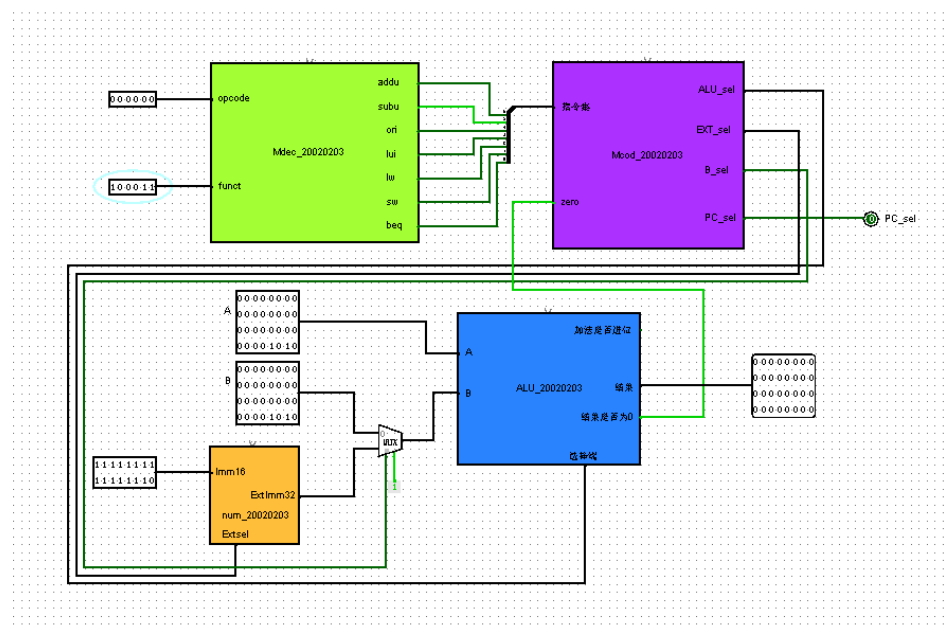
A	二进制 A	B	二进制 B	Imm16	二进制 Imm16	opcode	funct	功能	结果
32	00100000	-30	11100010	-6	11111010	000000	100001	addu	
10	00001010	10	00001010	-2	11111110	000000	100011	subu	
10	00001010	10	00001010	-1	11111111	001101	x	ori	

0	00000000	10	00001010	15	00001111	001111	x	lui	
33	00100001	33	00100001	32	00100000	000100	x	beq	
2	00000010	10	00001010	32	00100000	000100	x	beq	
10	00001010	8	00001000	-5	11111011	100011	x	lw	

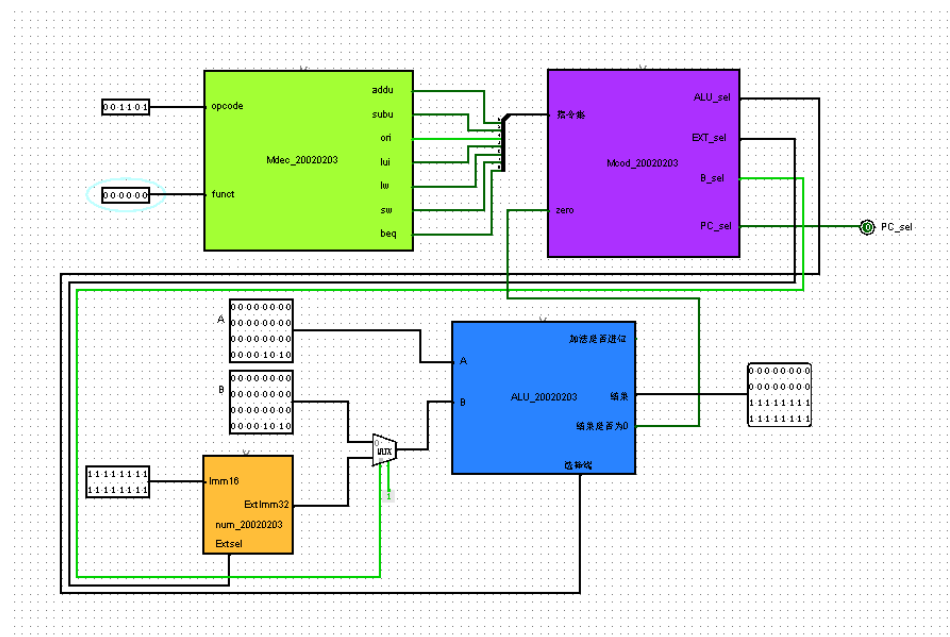
(1) 当 A=32、B= -30、Imm16=-6 时，完成 addu 测试，截取测试结果并说明；



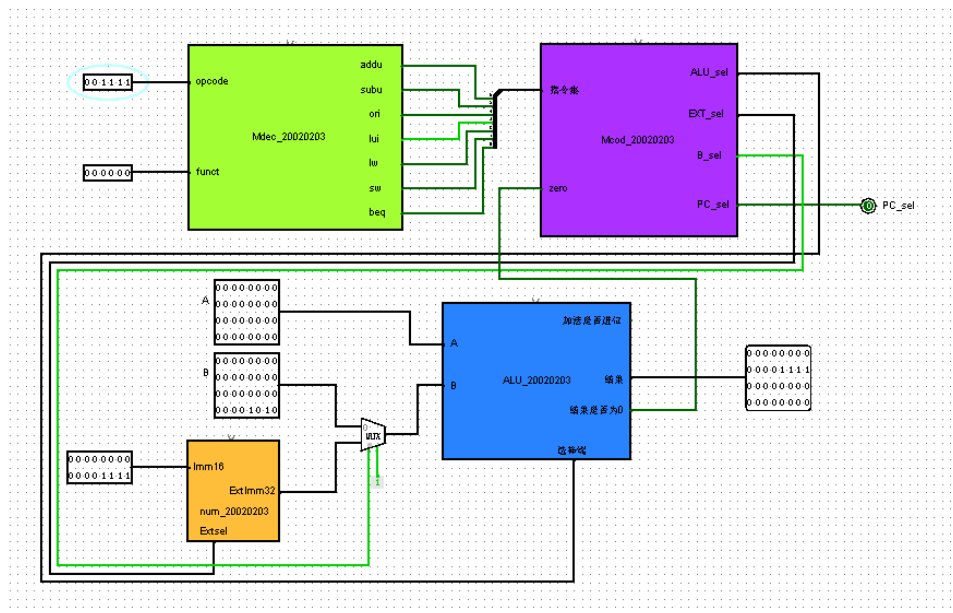
(2)当 A=10、B=10、Imm16=-2 时，完成 subu 测试，截取测试结果并说明；



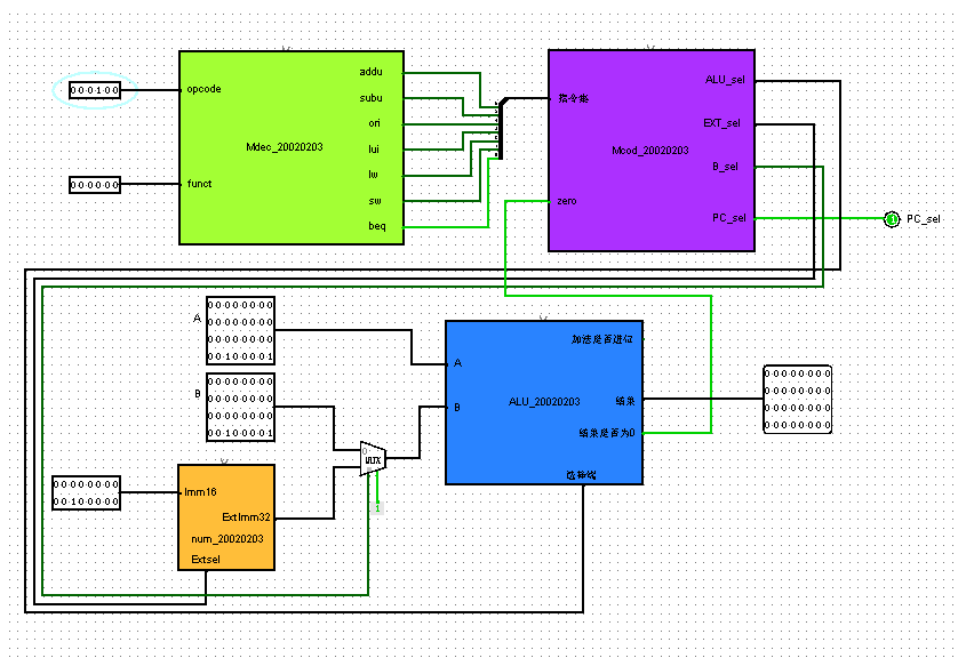
(3)当 A=10、B=10、Imm16=-1 时，完成 ori 测试，截取测试结果并说明；



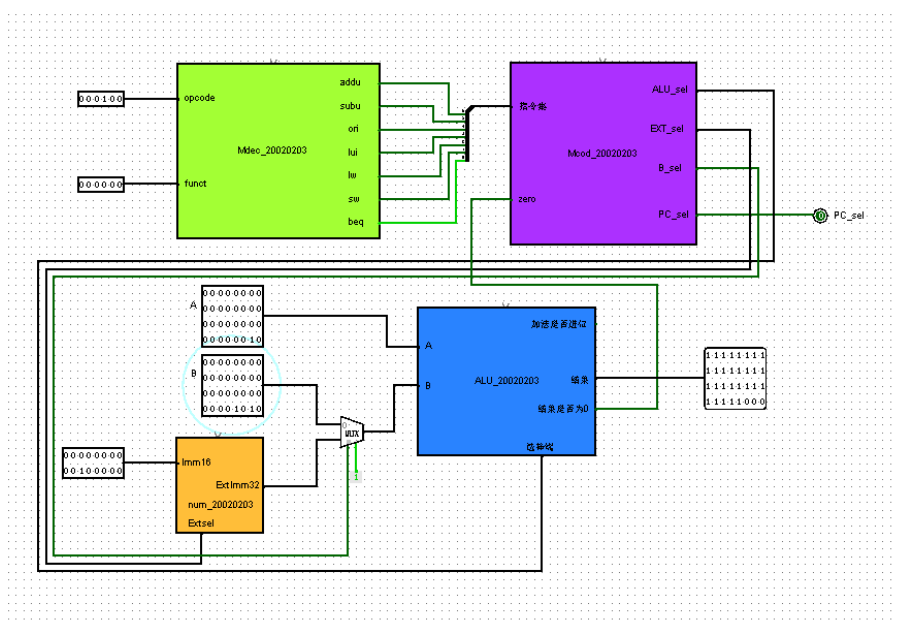
(4)当 A=0、B=10，Imm16=15 时，完成 lui 测试，截取测试结果并说明；



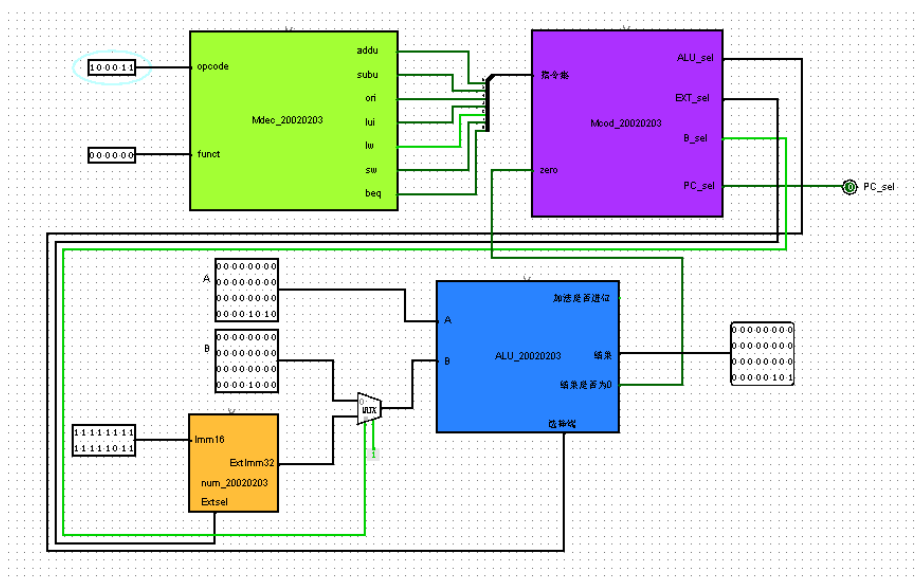
(5)当 A=33、B=33，Imm16=32 时，完成 beq 测试，截取测试结果并说明；



(6)当 A=2、B=10，Imm16=32 时，完成 beq 测试，截取测试结果并说明；



(7)当 A=10、B=8，Imm16=-5 时，完成 lw 测试，截取测试结果并说明；



七、综合以上内容，完成 WORD 报告汇集并进行小结。

本次大作业我按照要求完成了 ALU、MIPS、立即数扩展器、编码器等器件。32 位加法器、减法器是 logisim 的基本器件，但我们这次通过使用一位加法器不断串联，得到了自制的 32 位加法器，减法器可以通过加法器实现，原理是 $A-B=A+[B]$ 补码，故完成了加法器与减法器的自行设计。在设计过程中，当我使用 verilog 硬件描述语言实现 ALU 的时候我注意到，&和&&是不一样的，&是按位与，而&&是逻辑与，这是本次设计过程中记忆较为深刻的点。在设计后面几个器件的时候，我还学会了几个新的器件的使用，分别为：分线器、数据选择器、位数扩展器等，我认为最有用的是‘constant’常量这个器件，它可以给元件一个常量输入，这样就可以使某个输入始终保持一个固定的值，不受外界输入干预。最后按照要求将几个器件连起来时，让我的逻辑一下变得清晰起来，即这次设计主要就是分为运算模块 ALU，发送指令模块，译码模块和立即数扩展模块。当指令模块发出指令后，指令被译为对应的选

择信号，这些选择信号控制着输入和 ALU 的运算两个部分，最后经 ALU 运算输出对应结果。

通过这次实验，我不仅更加熟练的掌握了 logisim 软件的使用，而且明白了简易“CPU”的结构、工作原理，这让我思维更加清晰，对数字逻辑这门课也有了更深的认识，真正做到了将学习的知识用到实际生活当中去。