一、模块设计

1. IFU

IFU为取指单元，根据PC从IM中获取指令。内部的两个主要功能部件是PC，NPC和IM。
NPC用于更新PC的值，IM用于存储要执行的指令并从中读取指令。

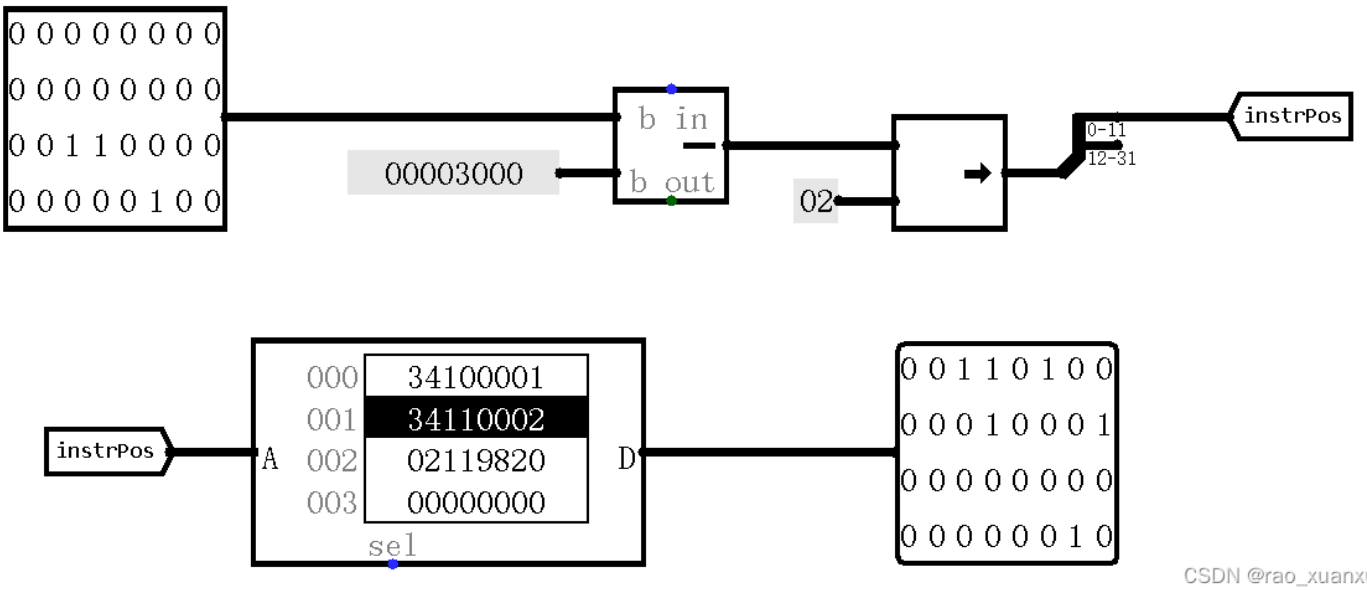
IM

对于IM，需要在电路运行前导入指令集。在CPU运行过程需要从IM读取指令，不需要写入指令，因此适合使用**ROM**这一部件实现。在MIPS指令集中，每条指令的长度为32bit，需要4个地址存储，但是在**ROM**中，设置 `Data Bit Width = 32` ，那么每条指令只需要1个地址存储。MIPS中，指令存储的地址范围是0x00003000到0x00006FFF，做如下映射即可将PC转化为指令存储在ROM的地址编号pos：$\text{pos} = (\text{PC} - 0\text{x}00003000) >> 2$（ROM设置 `Address Bit Width = 12` ）。

IM的端口定义如下：

信号名	方向
PC[31:0]	I
Instr[31:0]	O

在logisim中搭建电路：



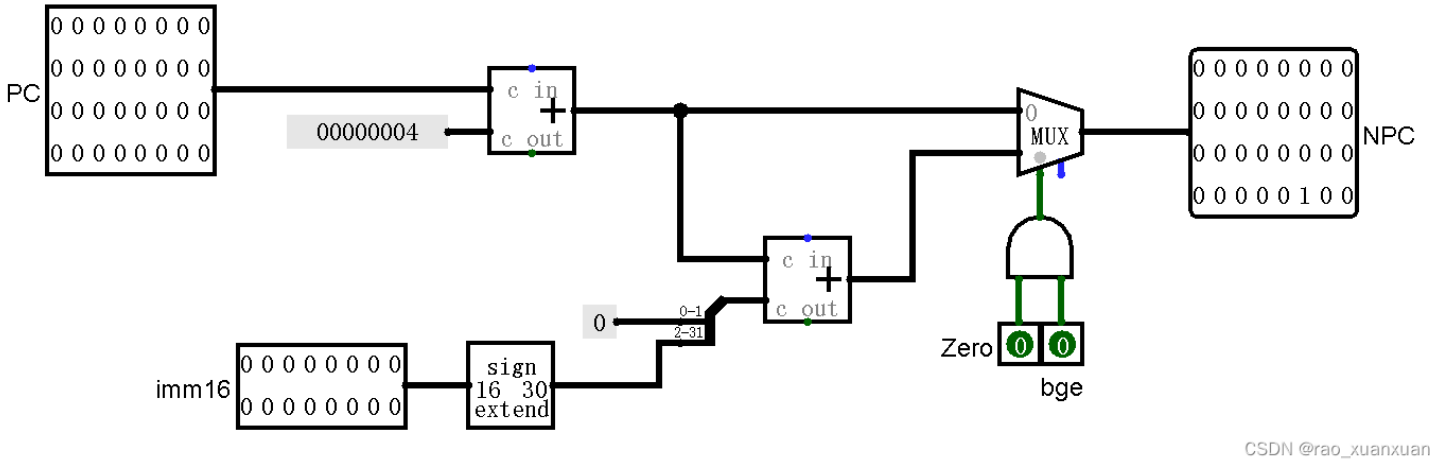
NPC

NPC用与计算下一条指令的地址，在我们所需要支持的指令集中，$\{\text{add, sub, ori, lw, sw, lui, nop}\}$这些指令执行完后应该顺序执行下一条，即PC <= PC+4；而$\{\text{beq}\}$则会在满足相等条件时跳转，PC <= PC+4+sign_extend(imm)||00，在不相等时顺序执行下一条指令，即PC <= PC+4。

NPC端口定义如下：

信号名	方向	说明
PC[31:0]	I	当前指令的地址
imm[15:0]	I	立即数
Zero	I	相等条件
Br	I	是否为分支跳转
NPC[31:0]	O	下一条指令的地址

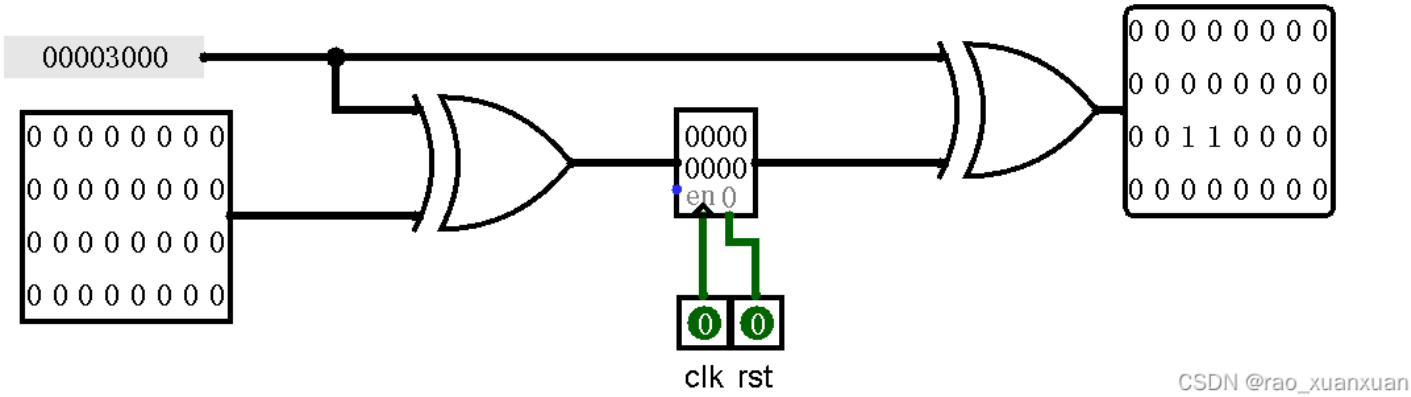
在logisim中搭建电路：



PC寄存器

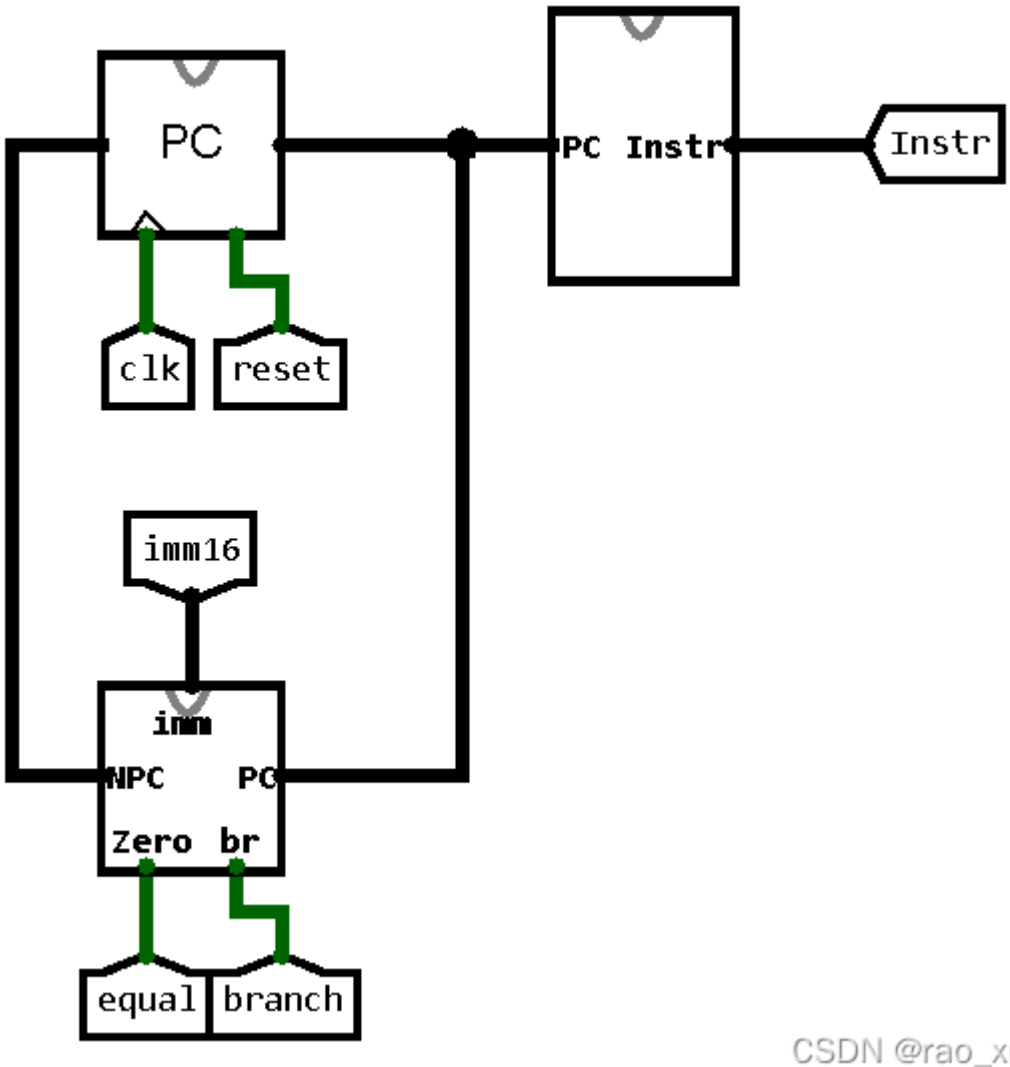
PC寄存器用于进行状态转移，输入为NPC，输出为PC，功能是在时钟上升沿时执行新的指令，直接使用logisim自带的register实现即可。

值得注意的是，每次对系统进行复位，PC应该回退到第一条指令的地址即0x00003000，但是logisim提供的register复位只能回到0，因此需要做出修改，保证每次复位可以回到指定值，电路如下：



IFU结构

结合IM、NPC和PC寄存器，可以得到IFU单元结构：

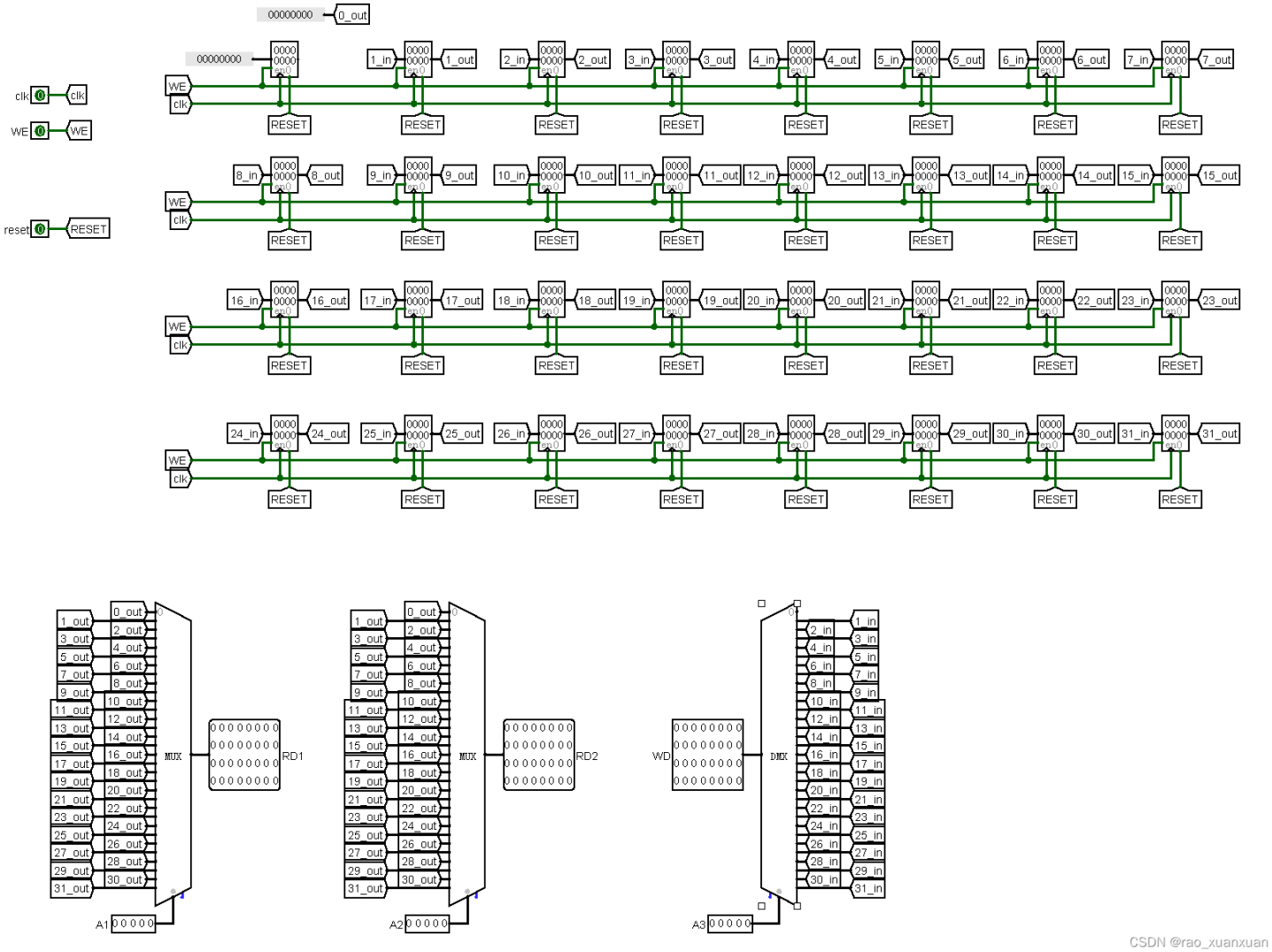


IFU单元类似于一个Moore型状态机，NPC进行状态转移，IM进行输出，而PC寄存器用于存储状态。

2. GRF

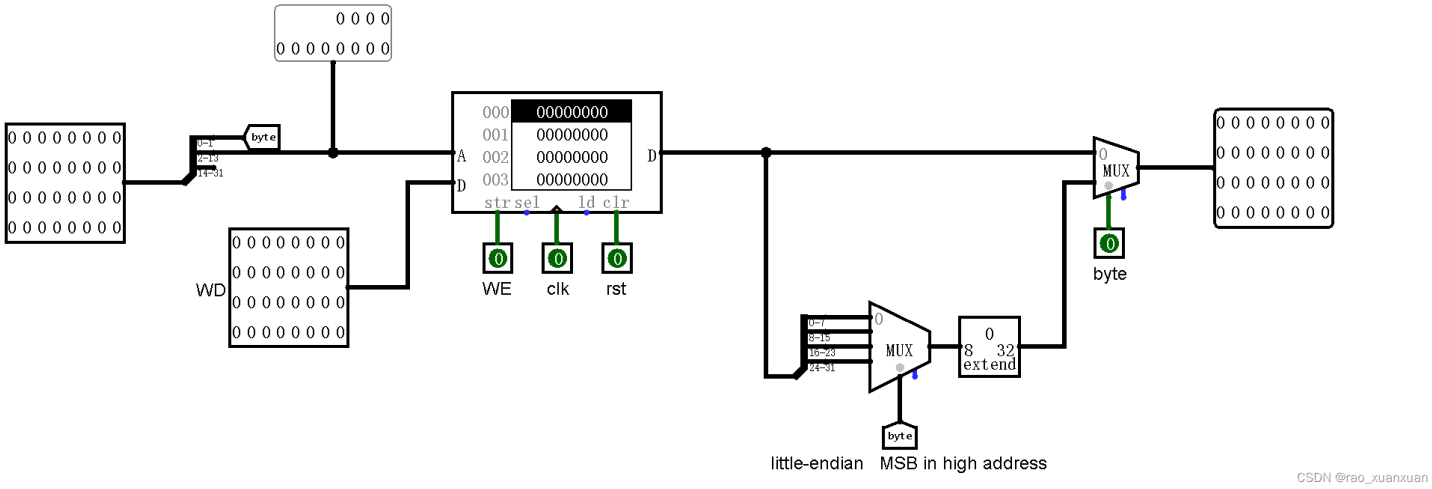
在CPU中，GRF为含有32个Register的寄存器阵列，用于存储0－31这32个寄存器的值。在CPU执行过程中，需要有读和写的功能：读取地址编号为A1和A2的寄存器内存储的值RD1和RD2，在WE信号有效时在时钟上升沿向地址编号为A3的寄存器写入数据WD。

读取数据的操作可以由多路选择器完成，以0－31这三十二个寄存器的值为输入，A为选择信号，RD为输出。写入数据的操作可以由Demultiplexer完成，以WD为输入，A为选择信号，0－31这三十二个寄存器的输入为输出，需要注意的是Demultiplexer需要设置 `Three-State = Yes` 。由于0寄存器的值始终为0，所以我们不能对0进行写入。



3. DM

DM相当于主存，用于存储数据，访问速度比寄存器慢，但是可以存储的数据多，需要进行读写，因此适合用RAM实现，设置 `Data Interface = separate load and store ports` 。在MIPS指令集中，数据按字存储和访问，需要设置RAM `Data Bit Width = 32` ，即每个字在RAM需要一个地址存储，由于数据地址范围：0x00000000 ~ 0x00002FFF，考虑指令**lw(sw)**，如果我们需要访问MIPS中地址为Addr的字，那么即访问RAM中地址为$\mathrm{Addr}>>2$的数据，因此设置RAM `Address Bit Width = 12` 。考虑指令**lb(sw)**，访问一个字节，那么只需要先取出字，使用Splitter分为4个字节，然后根据Addr的最低2位使用多路选择器根据大小端要求选出需要的字节。

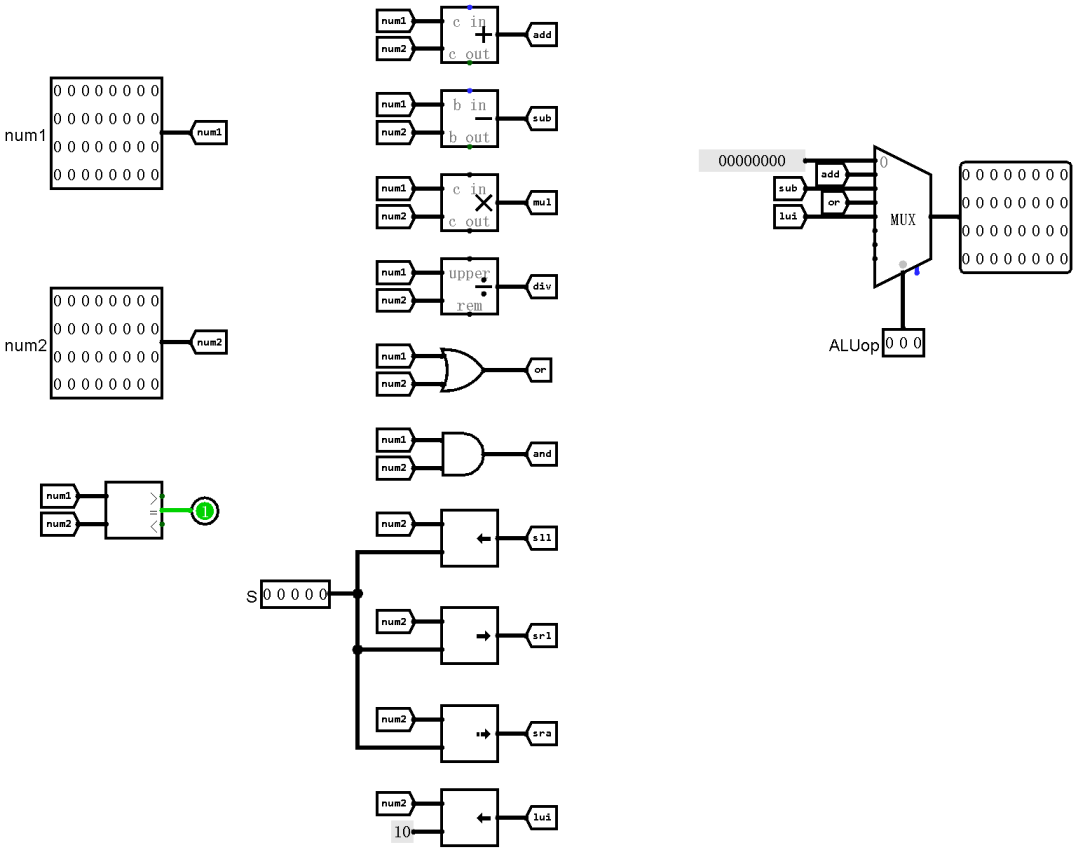


4. ALU

在指令执行过程中的运算过程由ALU完成，ALU有两个32位操作数srcA、srcB，可以进行add、sub、or、shift等操作，根据需要实现的功能，我们可以向ALU中加入新的功能，最后使用opCode信号选择输出。特别地，ALU需要设置**zero**信号，其意义为srcA和srcB是否相等，相等时zero=1。

ALUop与执行操作的对应关系：

ALUop	操作	说明
0	nop	do nothing
1	add	srcA + srcB
2	sub	srcA - srcB
3	or	srcA | srcB
4	lui	$\mathrm{srcB}_{15:0}\|\|0^{16}$



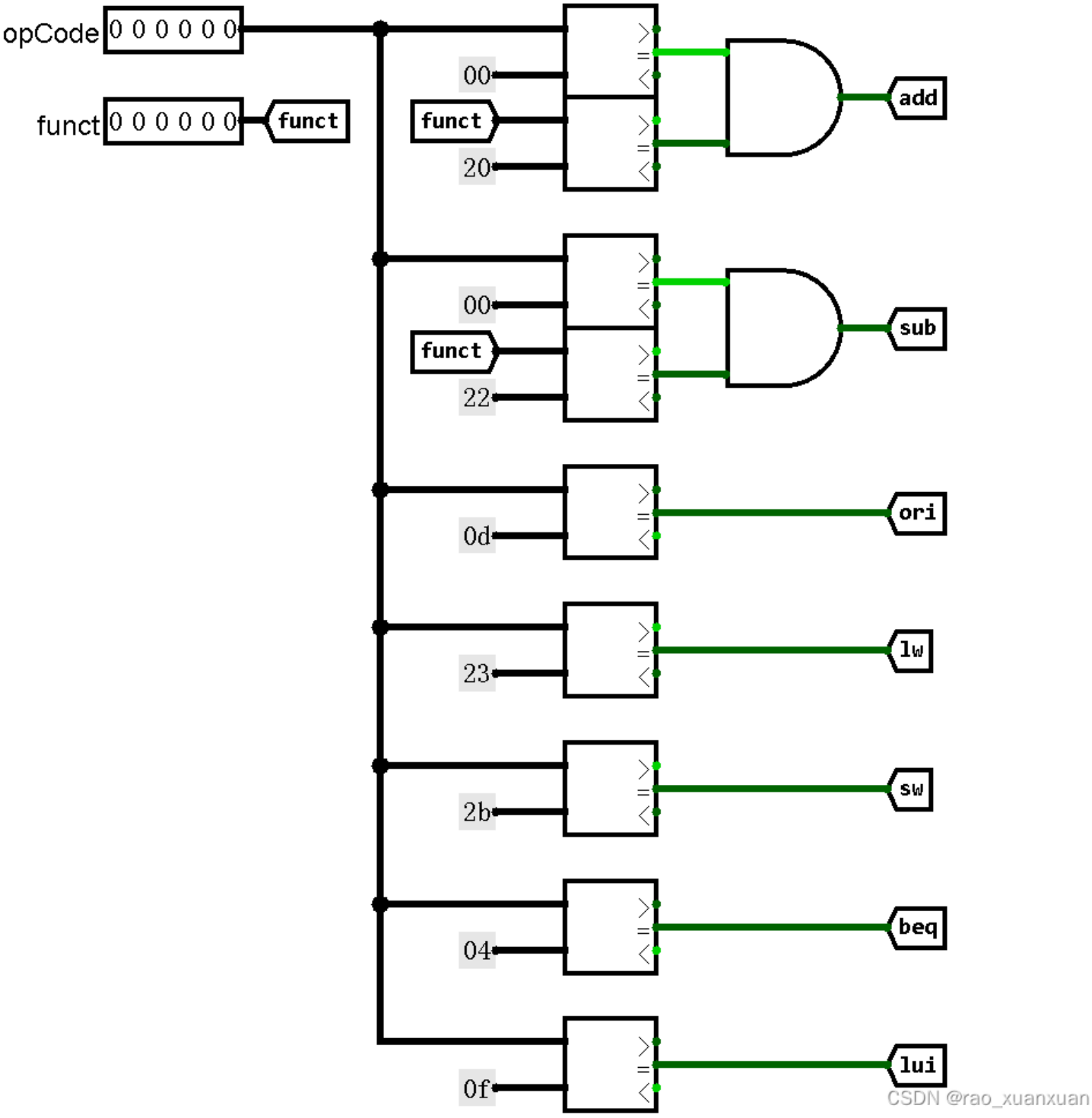
为了方便扩展移位操作指令，需要为ALU增加一个输入$\mathrm{shamt}[4:0]$，表示移动位数。

5. Controller

识别指令

对于一条MIPS指令，可以由opCode[31:16]和funct[5:0]这两部分得到，对于R型指令，opCode部分全为0，由funct决定类型，对于I型指令和J型指令，由opCode决定其类型。

可以用与门阵列或者比较器实现指令的识别，为了简化电路，采用比较器的方法识别指令。



生成控制信号

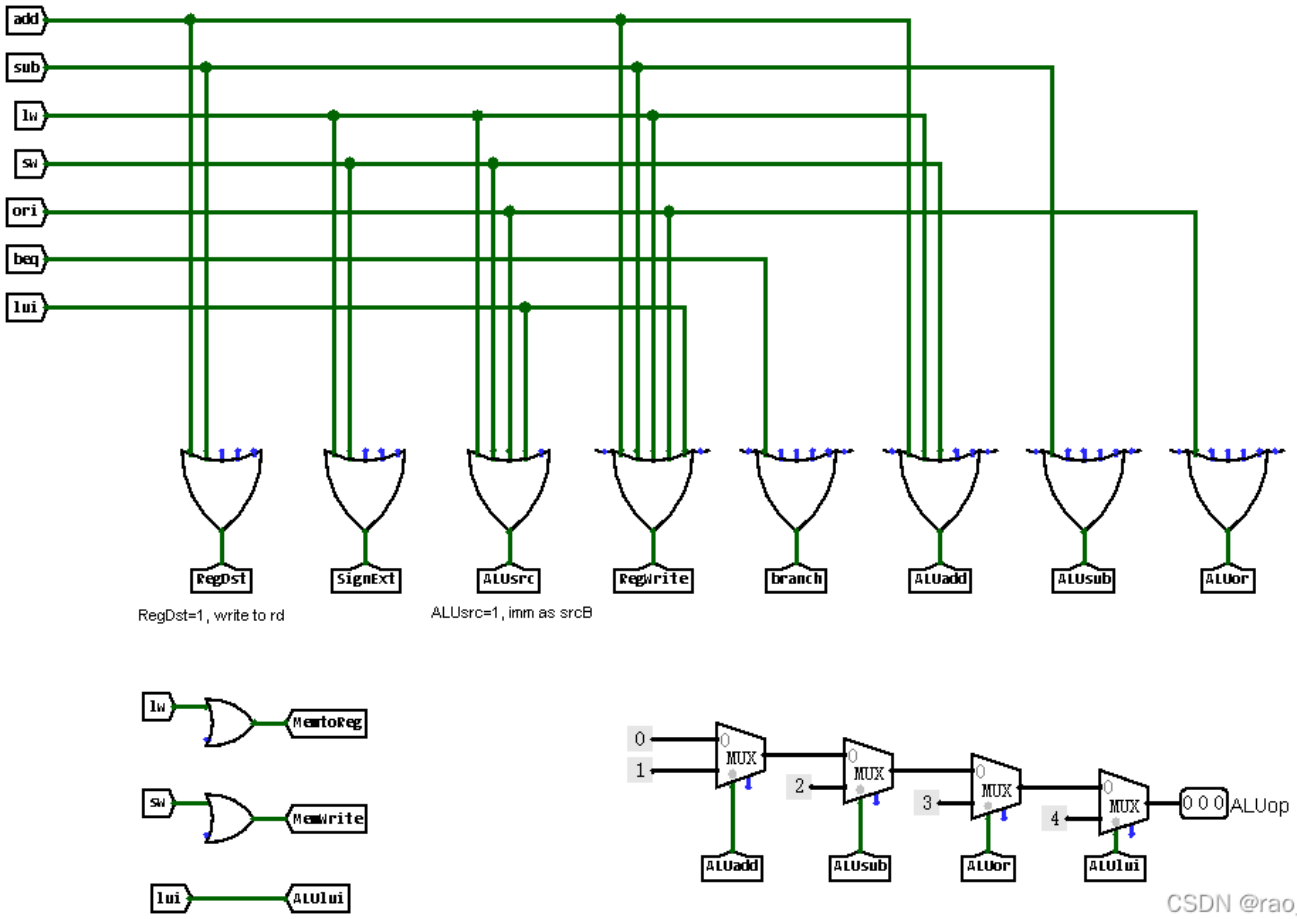
根据RF、ALU、DM在执行不同指令时的输入端连接方式，可以得到下表，然后根据表格生成控制信号。

指令	RF				ALU			DM	
	A1	A2	A3	WD	srcA	srcB	ALUop	A	WD
add	rs	rt	rd	ALU.add	RF.RD1	RF.RD2	add		
sub	rs	rt	rd	ALU.sub	RF.RD1	RF.RD2	sub		
ori	rs		rt	ALU.or	RF.RD1	zero_ext(imm16)	or		
lui	rs		rt	ALU.lui	RF.RD1	zero_ext(imm16)	add		
lw	rs		rt	MD.RD	RF.RD1	sign_ext(imm16)	add	ALU.add	
sw	rs	rt			RF.RD1	sign_ext(imm16)	add	ALU.add	RF.RD2
beq	rs	rt			RF.RD1	RF.RD2			

CSDN @rao_xuanxuan

控制信号可以分为对ALU操作类型相关的ALUop、影响数据通路的选择信号、决定是否写入数据的使能信号。根据连接关系和数据通路，可以得到各条指令Controller需要释放的控制信号。

指令	ALUop	RegDst	RegWrite	Extend	MemtoReg	MemWrite	branch
add	add	rd	1	\	0	0	0
sub	sub	rd	1	\	0	0	0
ori	or	rt	1	zero	0	0	0
lui	lui	rt	1	\	0	0	0
lw	add	rt	1	sign	1	0	0
sw	add	0	\	sign	0	1	0
beq	\	\	\	\	0	0	1



CSDN @rao_xuanxuan

二、测试方案

在Mars中编写汇编代码，运行得到结果，导出后加载到IFU的IM模块中，进行测试，运行后将logisim中GRF、DM的数据和Mars中的数据做比对。

测试代码如下：

```
ori $gp,$zero,0
ori $sp,$zero,0
ori $at,$zero,13398
add $at,$at,$at
lw $at,4($zero)
sw $at,4($zero)
lui $v0,30840
sub $v1,$v0,$at
lui $a1,4660
ori $a0,$zero,5
nop
sw $a1,-1($a0)
lw $v1,-1($a0)
beq $v1,$a1,tag1
beq $zero,$zero,tag2
tag1:
ori $a3,$v1,1028
beq $a3,$v1,tag3
nop
lui $t0,30583
ori $t0,$t0,65535
sub $zero,$zero,$t0
ori $zero,$zero,4352
add $t2,$a3,$a2
ori $t0,$zero,0
ori $t1,$zero,1
ori $t2,$zero,1
tag4:
add $t0,$t0,$t2
beq $t0,$t1,tag4
tag2:
tag3:
tag5:
beq $zero,$zero,tag5
```

导出为16进制机器码：

```
v2.0 raw
341c0000
341d0000
34013456
00210820
8c010004
ac010004
3c027878
00411822
3c051234
34040005
00000000
ac85ffff
8c83ffff
10650001
1000000d
34670404
10e3000b
00000000
3c087777
3508ffff
00080022
34001100
00e65020
34080000
34090001
340a0001
010a4020
1109fffe
1000ffff
```

三、思考题

1. 在组成单周期CPU的模块中，发挥状态存储功能的有PC寄存器、GRF，发挥状态转移功能的有NPC、ALU、Controller
2. IM在运行前需要导入待执行的指令，在CPU运行时，根据PC值从中取出指令，不需要向其中写入指令，因此适合选择**ROM**。
   DM在CPU运行时需要完成两个功能：从memory中读取地址处的值，向给定地址写入内容修改memory，有read和write两个功能，memory需要存储大量数据并且访问频率不高，因此适合选择

RAM。

GRF内部是一个寄存器阵列，在执行过程中需要向其中读取和写入数据，并且CPU在运行过程几乎每条指令都需要访问GRF，因此需要使用**GRF**加快访问速度以提高CPU效率。

3. 设计模块完成PC到指令在ROM中地址的转化。将$(\mathrm{PC}-0\mathrm{x}00003000)>>2$作为当前PC对应指令在ROM的地址。
4. 不将nop加入控制信号真值表，那么对于nop指令，Controller释放的信号保证了IM、DM存储数据不变，相当于在一个周期内除了PC加4外CPU没有执行任何行为
5. 测试样例对于 `ori,lui,add` 指令进行了高覆盖的测试，包括在正负多种情况下的测试，但是 `sw,lw,beq` 指令的测试中立即数都为正数，并且没有对边界情况进行检测，比如对于**sw**指令可以尝试让offset为负值，更好地测试对于**sw**指令，是否存在符号拓展上的问题，或者对于**beq**让$\mathrm{pc}+4+\mathrm{sign_ext}(\mathrm{imm})$为接近指令地址最大值0x00006FFF的值，以测试符号扩展或者IFU单元是否正确。