CPU 设计文档

一、 模块规格

1. IFU（取指令单元）：内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑。

PC 用寄存器实现，具有复位功能。

起始地址：0x00000000。

IM 用 ROM 实现，容量为 32bit * 32。

因 IM 实际地址宽度仅为 5 位，故使用恰当的方法将 PC 中储存的地址同 IM 联系起来。

表 1 IFU

序号	Input/output	功能描述
1	reset	当复位信号有效时，PC 被设置为 0x00000000
2	clk	时钟信号
3	NEWpc	下一个 pc 值，在时钟上升沿来时存入 pc
4	pc	输出 pc 信号
5	out	输出 IM 中取出的指令

设计如图：

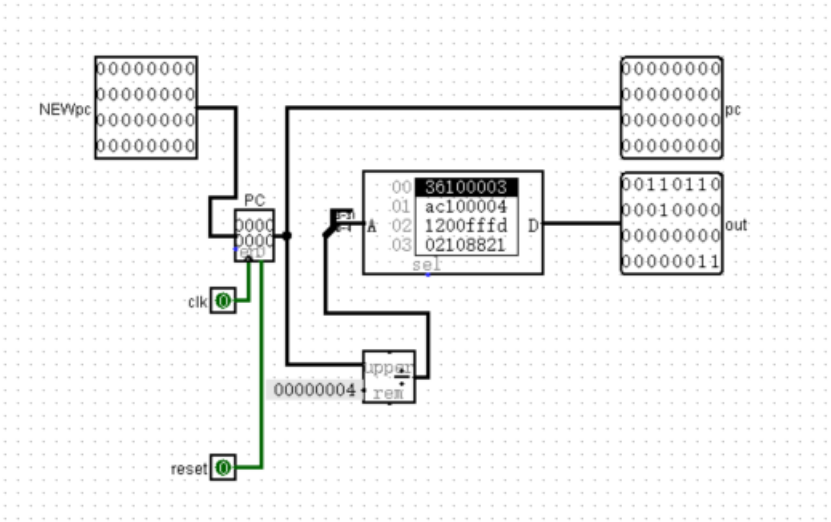

图 1.IFU 设计

2. GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

用具有写使能的寄存器实现，寄存器总数为 32 个。

0 号寄存器的值始终保持为 0,通过使其 clear 端始终为 1 实现。其他寄存器初始值均为 0，无需专门设置。

表 2 GRF

序号	Input/output	功能描述
1	reset	当复位信号有效时，PC 被设置为 0x00000000
2	clk	时钟信号
3	A1	读地址信号
4	A2	读地址信号
5	A3	写地址信号
6	WD	写使能信号
7	RD1	数据输出
8	RD2	数据输出

设计如图：

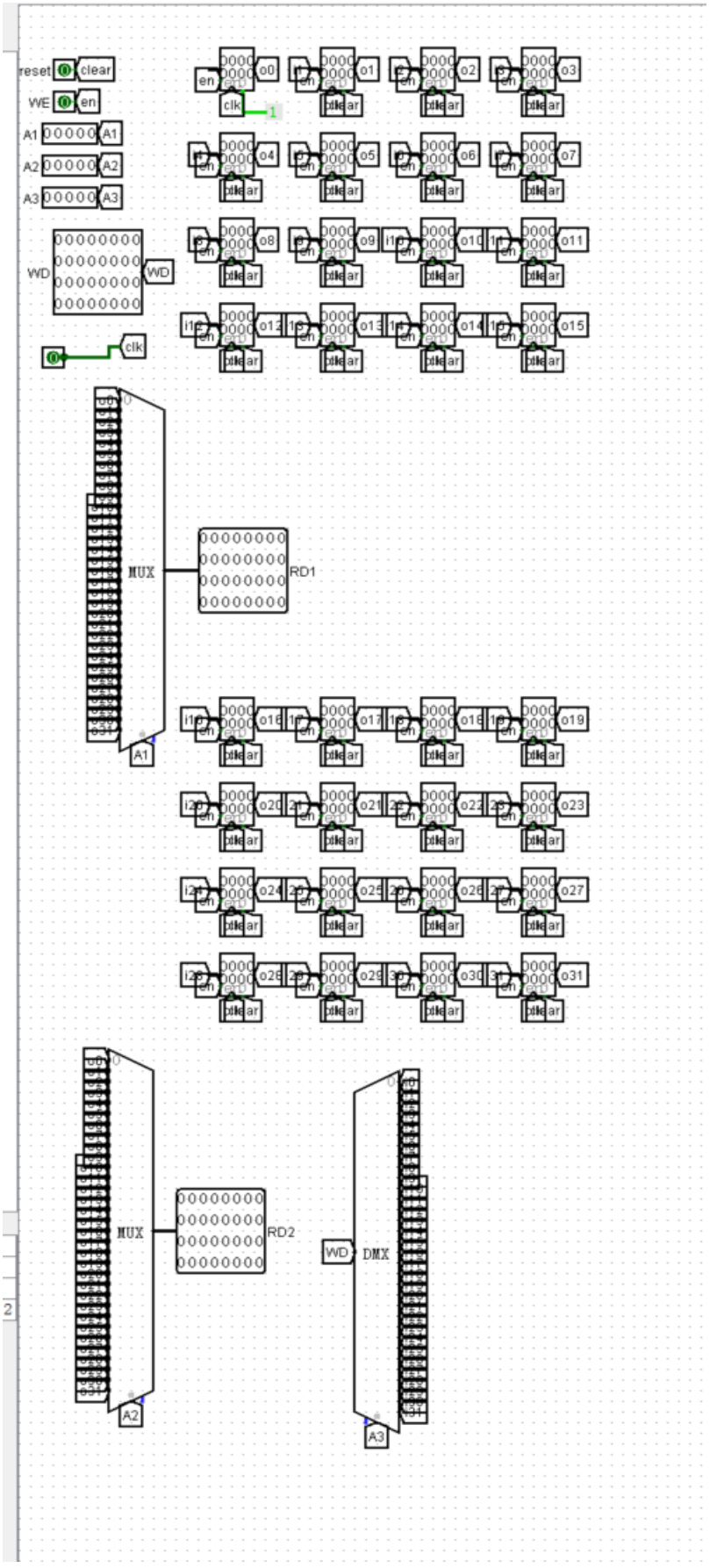

图 2.GRF

3. ALU（算术逻辑单元）

提供 32 位加、减、或运算及大小比较功能。

可以不支持溢出（不检测溢出）。

表 3 ALU

序号	Input/output	功能描述
1	In1	输入数据 1
2	In2	输入数据 2
3	ALUcontrol	选择输出，0000 选与运算，0001 选或运算，0010 选加法运算，0110 选减法运算
4	Out	输出计算结果
5	Zero	若两输入数据相等则输出 1，否则输出 0

设计如图：

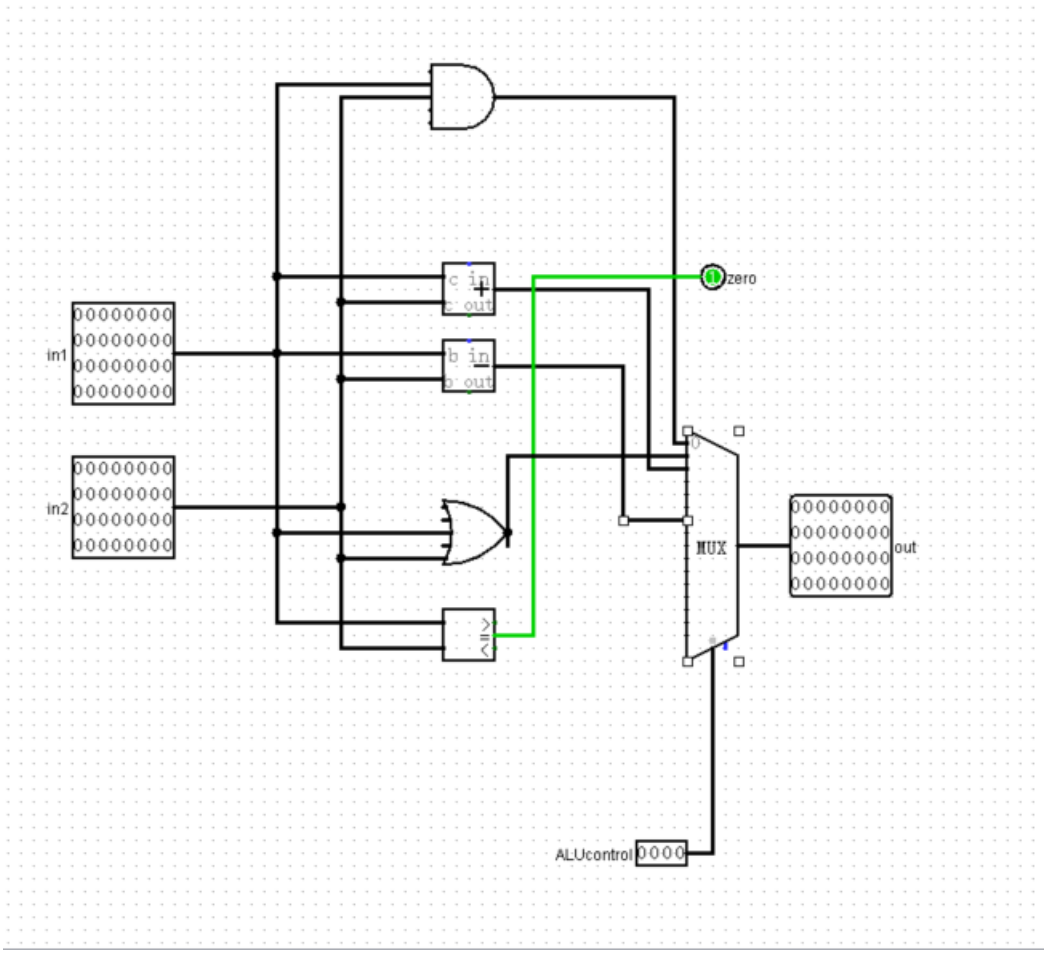

图 3.ALU

4. DM（数据存储器）

使用 RAM 实现，容量为 32bit * 32。

起始地址：0x00000000。

RAM 使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。

表 4 DM

序号	Input/output	功能描述
1	Add	访问地址
2	Data	输入数据
3	Write	写使能信号，当为 1 时，会在时钟上升沿存储数据
4	Read	读使能信号，从 RAM 中读出数据
5	Clk	时钟信号
6	Reset	重置信号，为 1 时清除 RAM 中所有数据
7	Select	选择信号，为 1 时选择从 RAM 中读出的信号作为输出，否则将输入 add 作为输出
8	Out	输出数据
9	Memadd	输出 RAM 被访问的地址

设计如图：

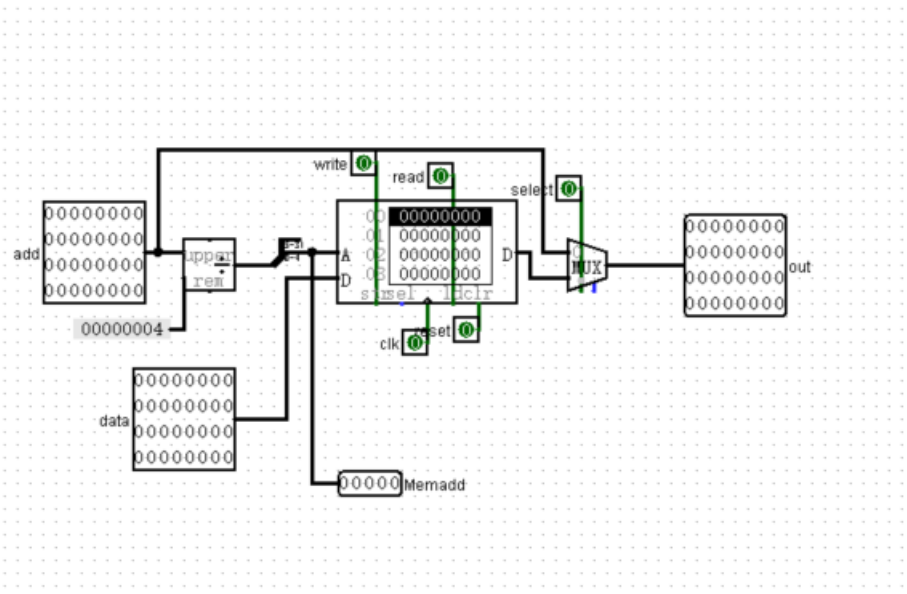

图 4.DM

5. EXT 以及 ALUController：

使用 logisim 内置的 Bit Extender

表 5 extender

序号	Input/output	功能描述
1	dataIn	输入指令码
2	Ori	当指令为 ori 时为 1，进行 0 扩展，否则进行符号扩展
3	Lui	当指令为 lui 时为 1，在低位扩展 16 个 0，否则进行其他扩展
4	Out	输出拓展后的立即数

表 6 ALUController

序号	Input/output	功能描述
1	Func	输入指令码后 6 位
2	ALUOp	连接 controller 中的输出信号 ALUOp
3	ALUout	输出 ALU 操作码

设计如图：

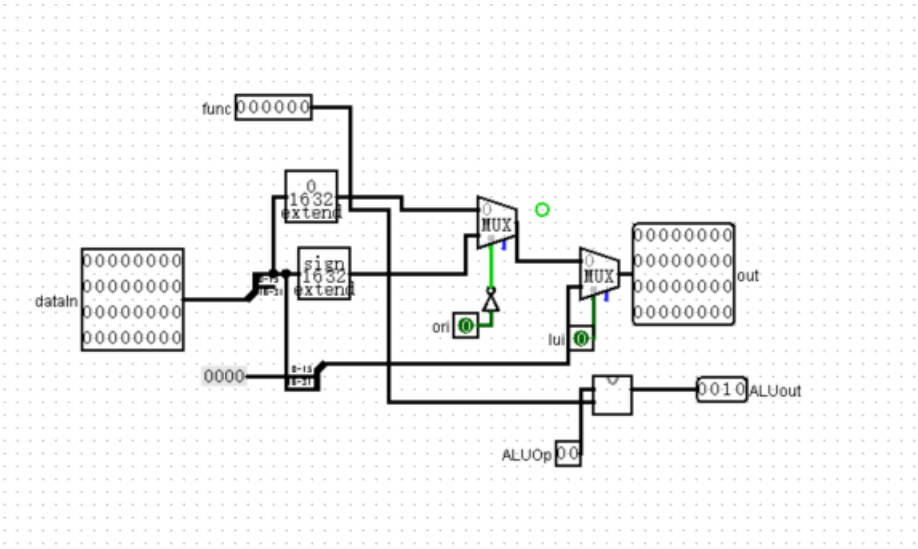

图 5.EXT

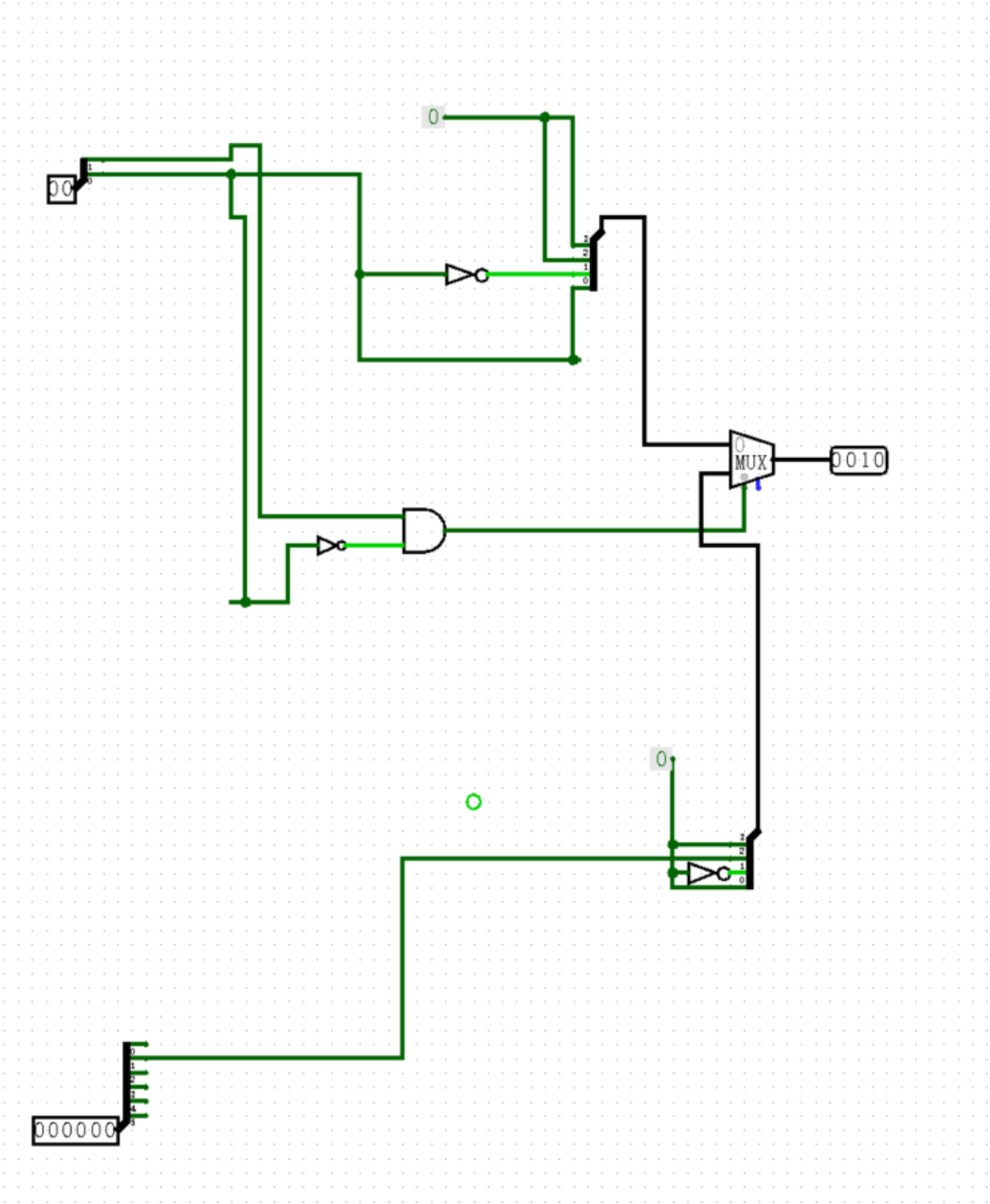

图 6.ALUController

二、 控制器设计

真值表：

	R-mat	Ori	Lw	Sw	Beq	Lui
RegDst	1	0	0	X	x	0
ALUSrc	0	1	1	1	0	1
MentoReg	0	0	1	X	x	0
RegWrite	1	1	1	0	0	1
MenRead	0	0	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
ALUOp1	1	0	0	0	0	0
ALUOp2	0	1	0	0	0	0
LUI	0	0	0	0	0	1

通过和逻辑识别指令码前六位：

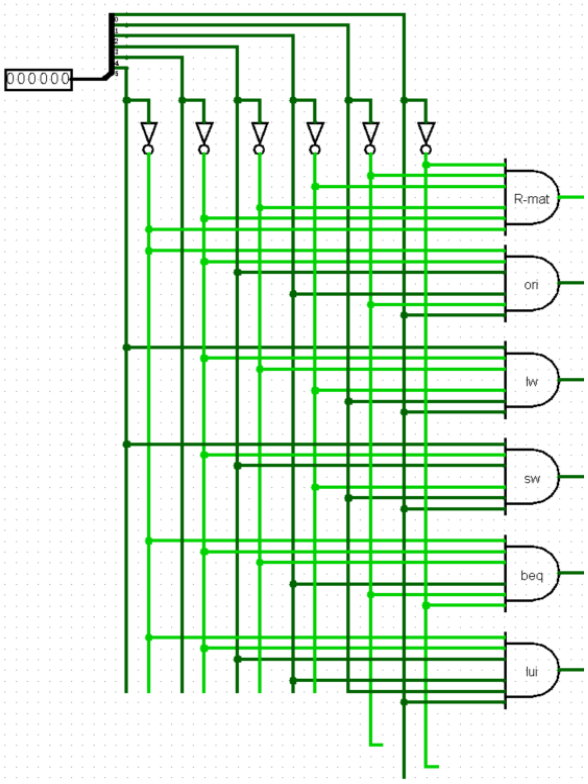

图 7.control-and

通过或逻辑生成输出信号：

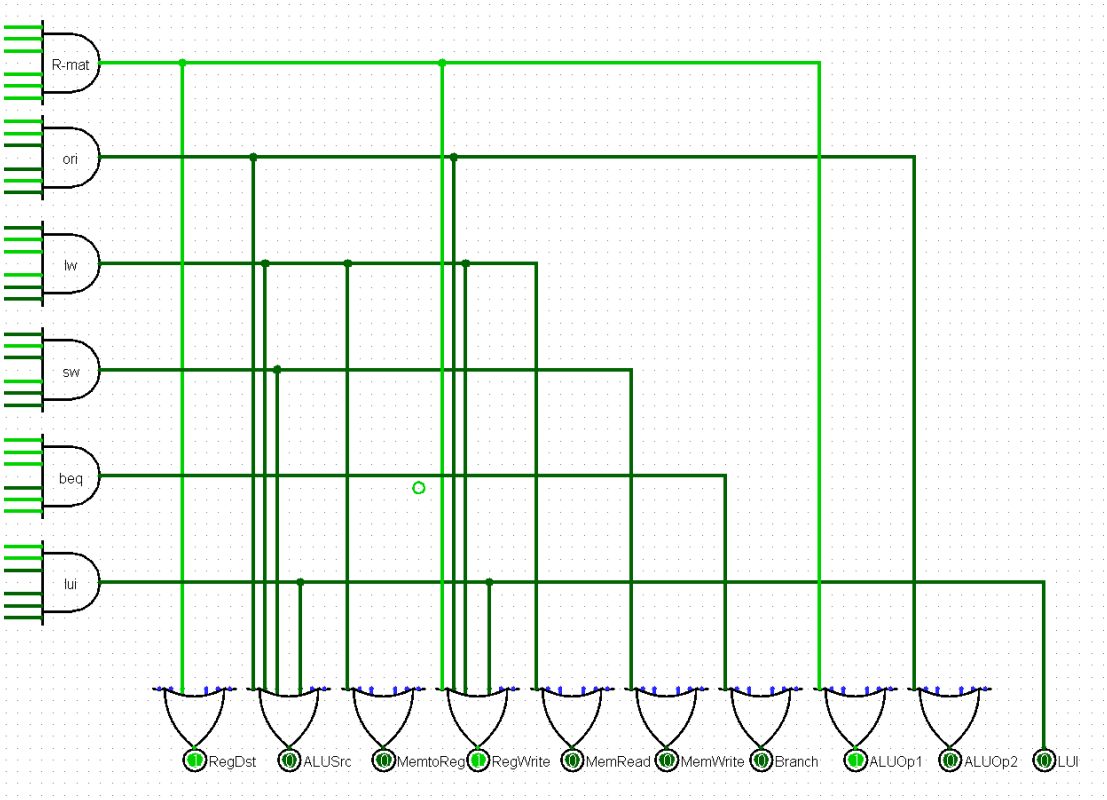

图 8.control-or

其中，lui、ALUOp2(ori)用于 extender 中对扩展方式的选择，可视为 ExtOp 信号。

ALUOp 指明 ALU 的运算类型：

00：访存指令所需的加法

01：ori 指令所需的或运算

10：R 型指令功能码字段决定

三、 测试程序

代码：

```
loca:
ori $s0,$s0,3
sw $s0,4($0)
nop
beq $s0,$0,loca
addu $s1,$s0,$s0
subu $s2,$s1,$s0
lw $s0,3($0)
beq $s0,$0,loca
```

机器码：v2.0 raw

```
36100003
ac100004
00000000
1200fffd
02108821
02309023
8c100003
1200fff9
```

期望输出：1.$s0 中为 3

2.RAM 中第二个数据为 3

3.无操作

4.不发生 branch

5.$s1 中为 6

6.$s2 中为 0

7. $s0 中为 0

8.发生 branch，回到第一步

四、 思考题

模块规格思考题：

1. 若 PC（程序计数器）位数为 30 位，范围缩小为原来的四分之一，但需要的存储器容量也更小
2. 现在我们的模块中 IM 使用 ROM，DM 使用 RAM，GRF 使用寄存器。合理。首先 grf 本来就是通用寄存器组。IM 使用 ROM 保证断电时数据不会丢失，从而保证 cpu 中的程序不会丢失。二、而 DM 使用 RAM 是需要频繁对 DM 进行读写操作，同时重置后数据丢失是可以接受的。

控制器设计思考题：

1. RegDst：~op0~op1~op2~op3~op4~op5

   ALUSrc:

   (~op0~op1op2op3~op4op5)+(op0~op1~op2~op3op4op5)+(op0~op1op2~op3op4op5)+(~op0~op1op2op3op4op5)

   MemtoReg: op0~op1~op2~op3op4op5

   RegWrite:(~op0~op1~op2~op3~op4~op5)+(~op0~op1op2op3~op4op5)+(op0~op1~op2~op3op4op5)+ (~op0~op1op2op3op4op5)

   nPC_Sel(branch): ~op0~op1~op2op3~op4~op5

   ExtOp:

   Ori: (~op0~op1op2op3~op4op5)

   Lui: ~op0~op1op2op3op4op5
2. RegDst：~op0~op1~op2~op3~op4~op5

   ALUSrc:

   (~op0~op1op2op3~op4op5)+(op0~op1~op2~op3op4op5)+(op0~op1op2~op3op4op5)+(~op0~op1op2op3op4op5)

   MemtoReg: op0~op1~op2~op3op4op5

   RegWrite:(~op0~op1~op2~op3~op4~op5)+(~op0~op1op2op3~op4op5)+(op0~op1~op2~op3op4op5)+ (~op0~op1op2op3op4op5)

   nPC_Sel(branch): ~op0~op1~op2op3~op4~op5

ExtOp:

Ori: (~op0~op1op2op3~op4op5)

Lui: ~op0~op1op2op3op4op5

3. 指令码前六位都是 0，会被识别为 R 型指令，rs, rt, rd 都是 $0, 对$0 操作不会产生任何影响，所以不需要将它加入控制信号真值表。

测试 cpu 思考题：

1. 配置指令存储器起始地址为 0 地址时，mars 中有.data base address 在 0x00002000 处，故将 DM 的 add 信号中第 19 位作为 DM 的片选信号即可。
2. 形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了 100%。
   形式验证可能非常快，效率高。但是，不能有效地验证电路的性能。
   模拟方法严重依赖与测试数据的选取，而且效率不高