2020年北航计组P3实验报告

冯张驰 19373573

一、CPU设计方案综述

（一）总体设计概述

本CPU为logisim实现的单周期MIPS - CPU，支持的指令集包含{addu, subu, ori, lw, sw, beq, lui, nop}。为了实现这些功能，CPU主要包含了IFU、GRF、ALU、DM、EXT、Controller六个模块。

（二）关键模块定义

1. GRF

GRF端口定义

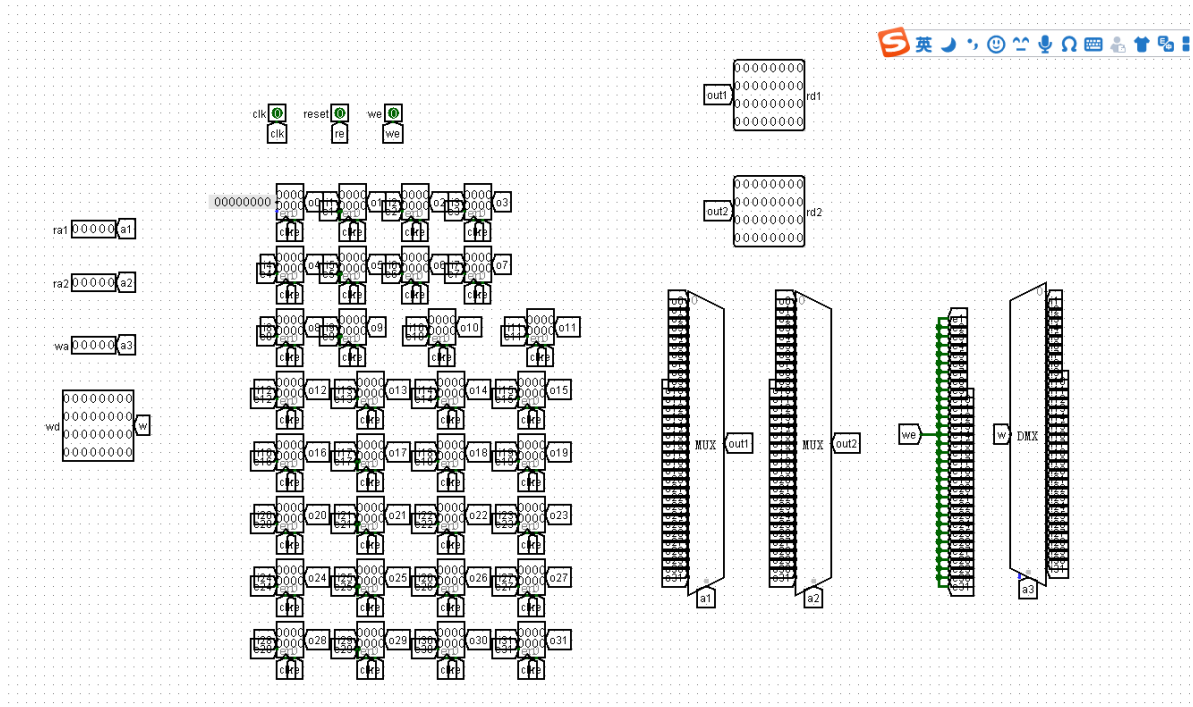
端口名	方向	描述
RA1[4:0]	I	RD1中数据的寄存器地址
RA2[4:0]	I	RD2中数据的寄存器地址
RD1[4:0]	O	RA1地址对应寄存器数值
RD2[4:0]	O	RA2地址对应寄存器数值
WA	I	写入数据的寄存器地址
WD	I	写入WA对应寄存器的数据
WE	I	写使能信号
clk	I	时钟信号
reset	I	异步复位信号

GRF功能定义

1.复位。复位信号有效时所有寄存器的数值被设置为0x00000000.

2.读寄存器 根据当前输入的地址信号读出32位数据。

3.写寄存器。根据当前输入的地址信号，把输入的数据写入相应寄存器。



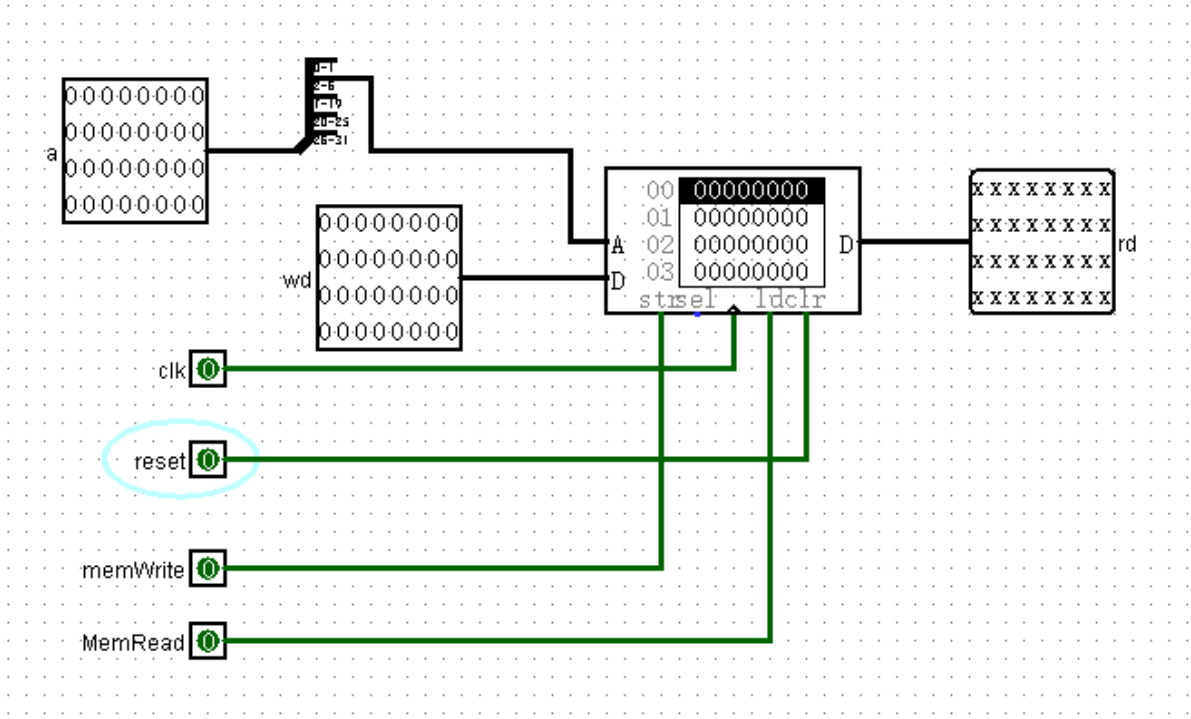
2. DM

DM端口定义

端口名	方向	描述
A[4:0]	I	数据的地址信号
WD[31:0]	I	当WE为高电平时，写入地址A的数据
RD[31:0]	O	当RE为高电平时，读出地址A的数据
RE	I	高电平时允许读数据
WE	I	高电平时允许写入数据
clk	O	时钟信号
reset	I	异步复位信号

DM模块功能定义

1. 复位：复位信号有效时，所有数据被设置为0x00000000。
2. 读：根据当前输入的寄存器地址读出数据
3. 写数据：根据输入地址，写入输入的数据。



3. ALU

ALU端口定义

端口名	方向	描述
A[31:0]	I	数据1
B[31:0]	I	数据2
aluCtrl[2:0]	I	控制信号000：与001：或010：加011：减101：比较大小
result	O	输出的数据
zero	O	判断A与B是否相等

功能定义

1.与：A&B

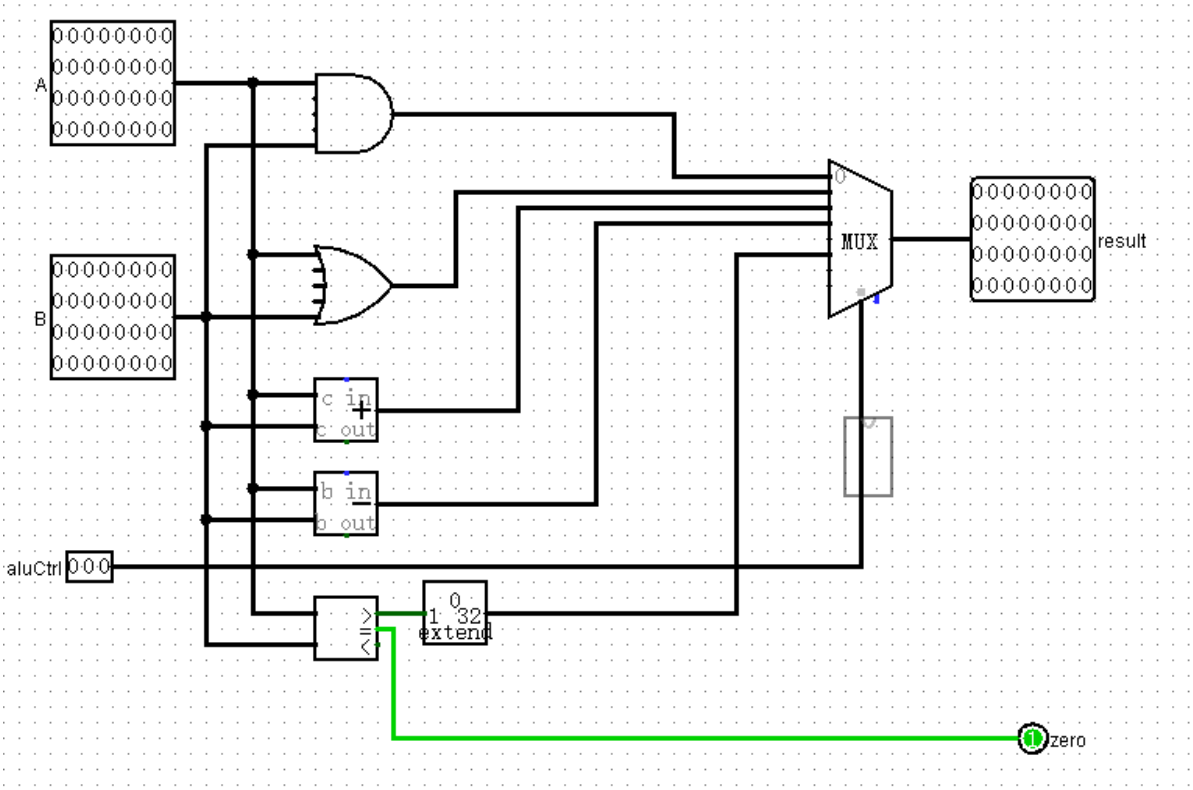
2.或：A|B

3.加：A+B

4.减：A-B

5.是否相等：A==B?

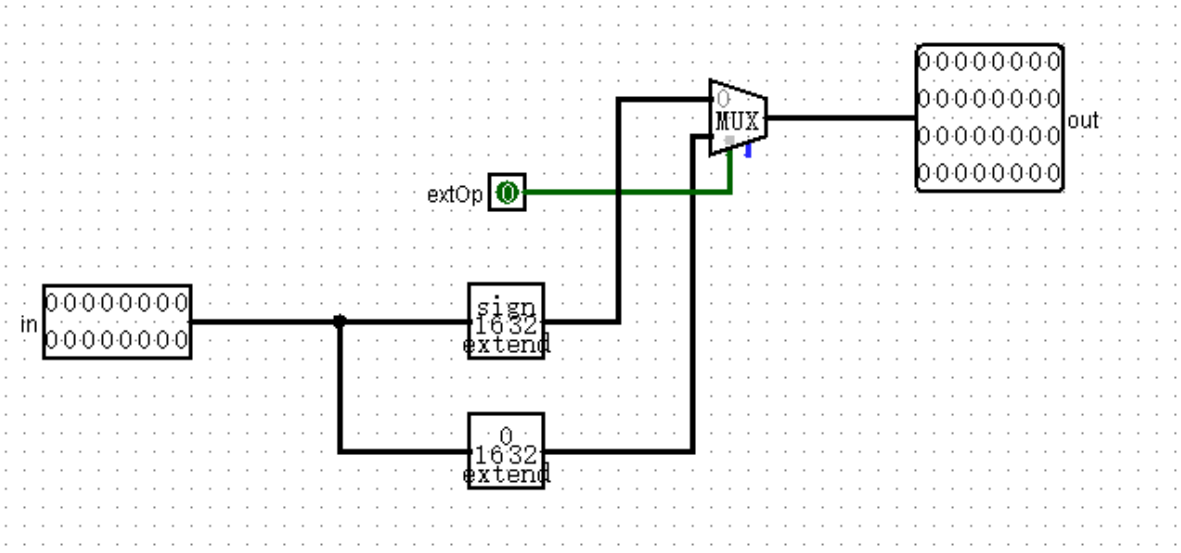
6.比较大小：A>B?



4. EXT

EXT端口定义

端口名	方向	描述
in[15:0]	I	16位立即数
extOp	I	扩展控制信号0：符号扩展1：无符号扩展
out[31:0]	O	扩展后输出的数据值



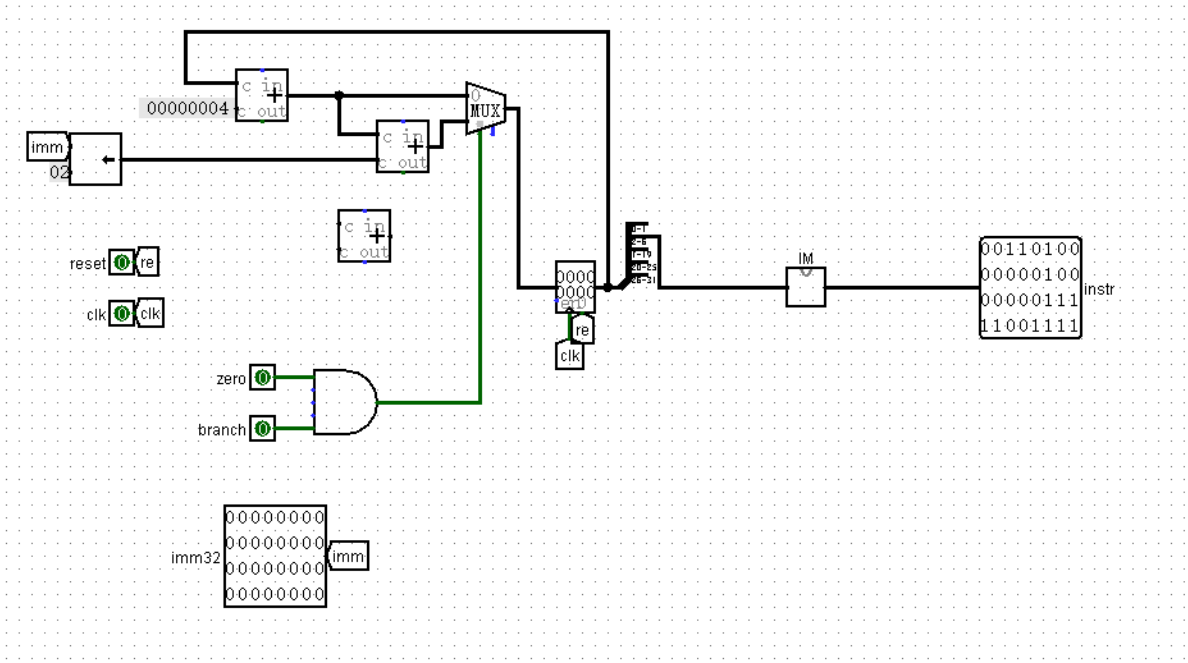
5.IFU

IFU端口定义

端口名	方向	描述
clk	I	时钟信号
reset	I	复位信号
zero	I	ALU计算结果是否为0
branch	I	当前指令是否为beq指令
instr	O	新的32位MIPS指令
imm[31:0]	I	32位地址（beq指令的立即数符号扩展后的数据）

IFU功能定义

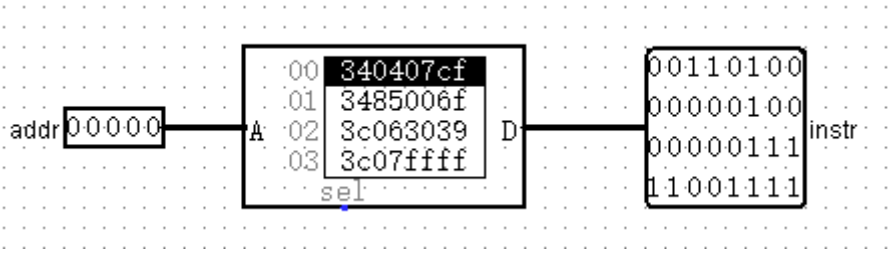
1. 复位 复位信号有效时，PC被设置为0x00000000
2. 取指令 根据PC当前值从IM中取出指令。
3. 计算下一条指令地址。如果当前指令不是beq指令，PC=PC+4；如果当前指令是beq指令且zero=1，则PC=PC+4+sign_extend(offset||02)



IFU中的子模块IM

IM端口定义

端口名	方向	描述
addr[4:0]	I	IFU中输入的5位地址信号
instr[31:0]	O	32位指令



6.Controller

端口定义

端口名	方向	描述
op[5:0]	I	6位opcode字段
regDst	O	写地址控制信号
aluSrc	O	ALU计算结果是否为0
regWrite	O	GRF写控制
memRead	O	DM读控制
memWrite	O	DM写控制
memToReg[1:0]	O	GRF写数据选择
extOp	O	高位扩展方式选择信号
branch	O	是否为beq指令
aluOp[2:0]	O	传递给AluCtrl模块，与func[5:0]共同确定aluCtrl[2:0]信号。

AluCtrl子模块端口定义

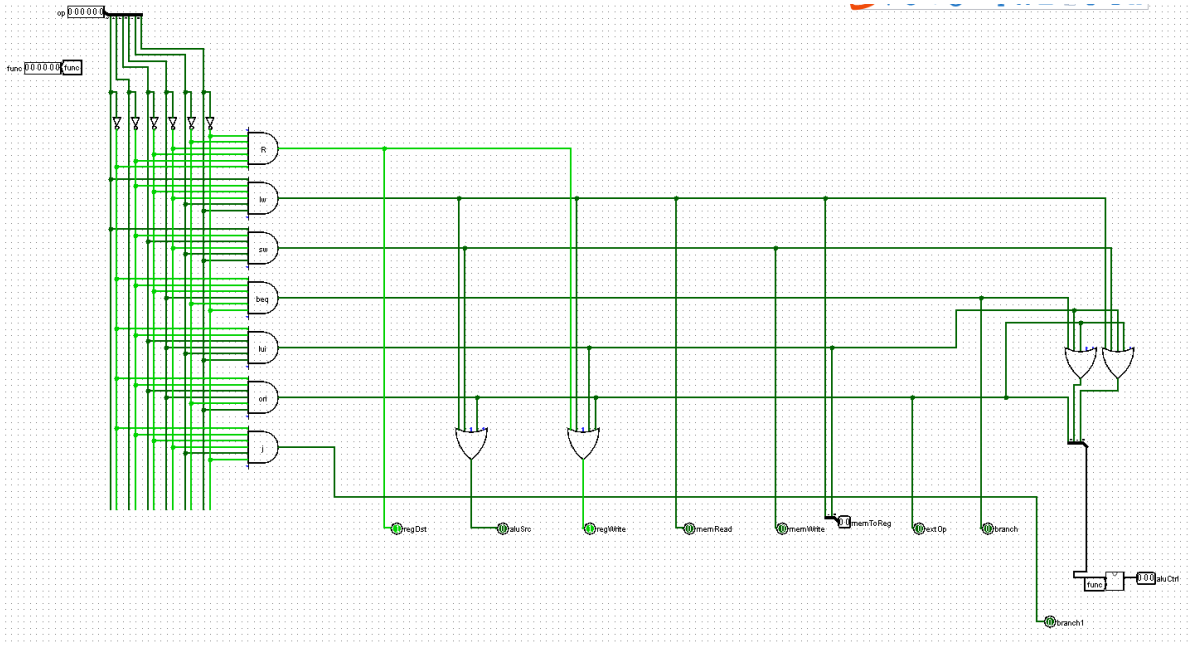
端口名	方向	描述
aluOp[2:0]	I	由op[5:0]产生的控制信号
func[5:0]	I	6位funcode信号
aluCtrl	O	ALU选择信号

（三）控制器设计思路

主控单元译码电路真值表

端口名字\输出信号	R（addu/subu)	lw	sw	beq	lui	ori	nop
op5	0	1	1	0	0	0	0
op4	0	0	0	0	0	0	0
op3	0	0	1	0	1	1	0
op2	0	0	0	1	1	1	0
op1	0	1	1	0	1	0	0
op0	0	1	1	0	1	1	0
regDst	1	0	x	x	0	0	x
aluSrc	0	1	1	0	x	1	x
regWrite	1	1	0	0	1	1	x
memRead	0	1	0	0	0	0	x
memWrite	0	0	1	0	0	0	x
memToReg[1:0]	00	10	xx	xx	01	00	xx
extOp	0	0	0	0	0	1	x
branch	0	0	0	1	0	0	x
aluOp[2:0]	000	001	001	010	011	111	xxx

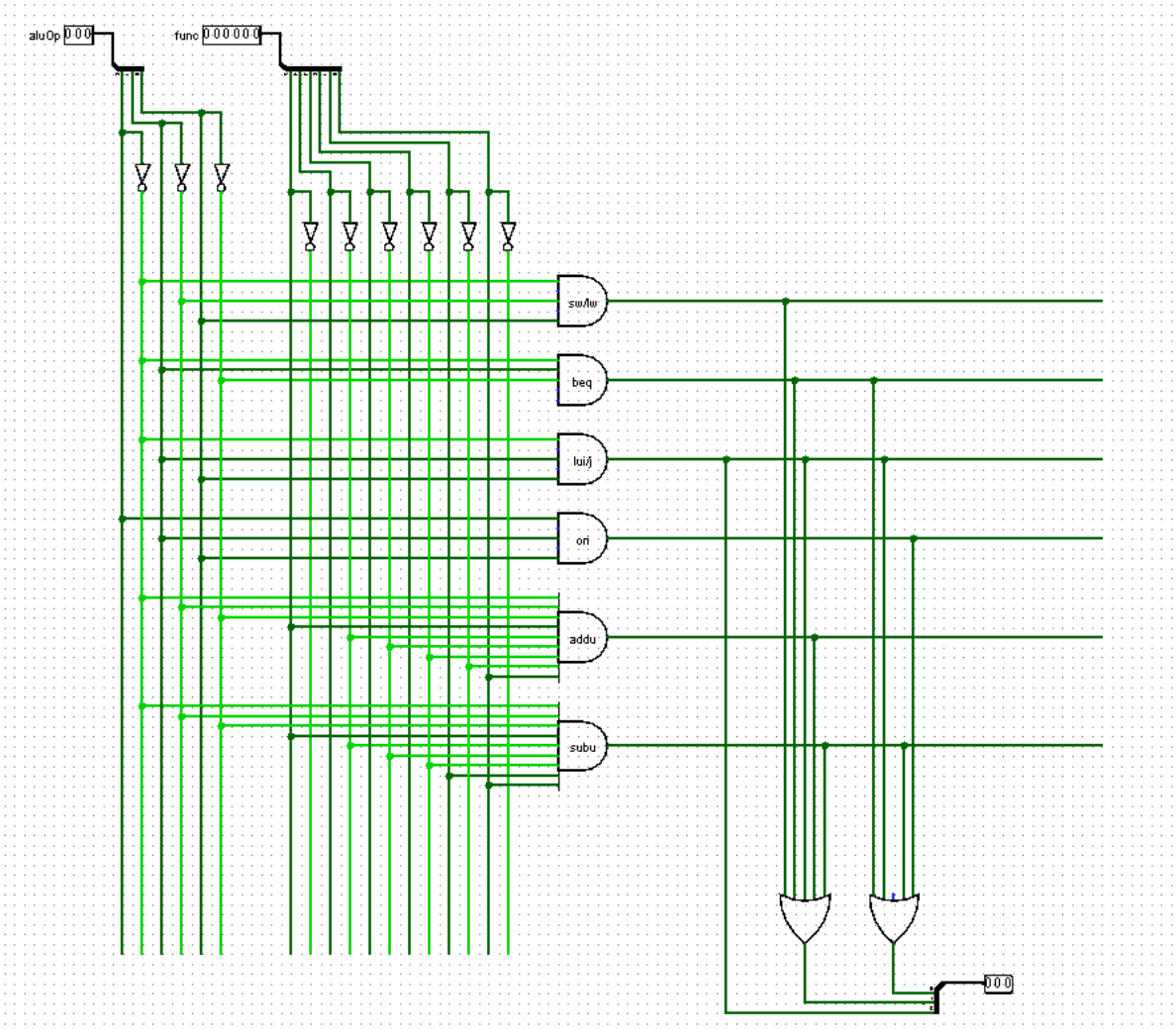
与或门阵列为：



运算控制单元译码真值表

指令\输入输出信号	func[5:0]	aluOp[2:0]	aluCtrl[2:0]
lw	xxxxxx	001	010
sw	xxxxxx	001	010
beq	xxxxxx	010	011
lui	xxxxxx	011	111
ori	xxxxxx	111	001
addu	100001	000	010
subu	100011	000	011

与或门阵列为：



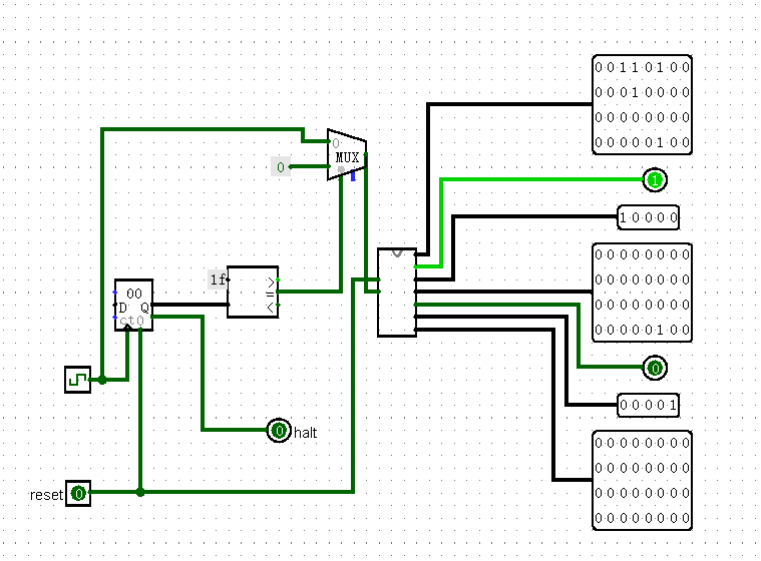
二、测试方案

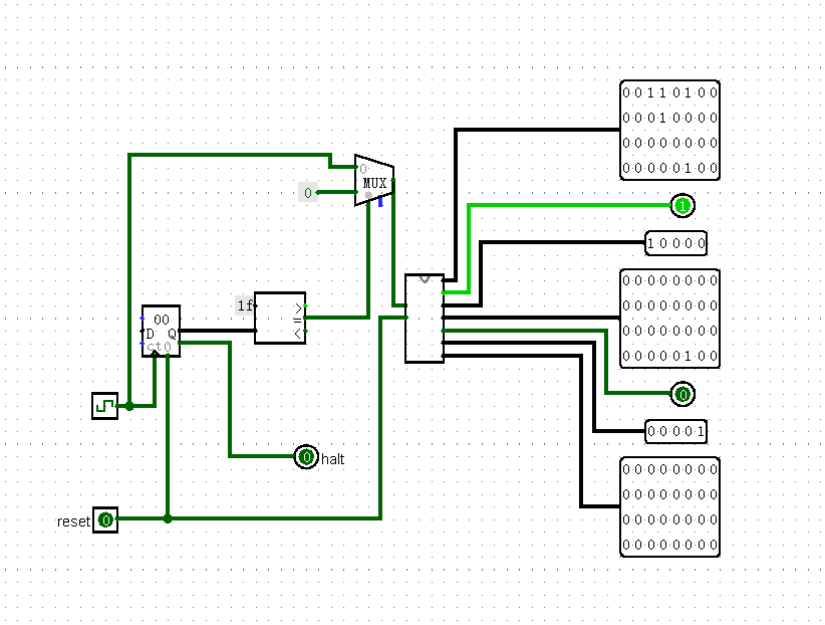
自动化测试脚本

基本方法描述

先随机生成一段除了`beq`指令之外的MIPS代码，之后利用`python`的`os`模块的相关函数使用命令行得到`Mars`的十六进制代码文件，然后将其写入`logisim`的`.circ`文件的`ROM`区域，最后和一个祖传的`logisim`电路文件进行基于`bash命令行`的`FC`对拍，得到结果。（以下代码参考了评论区一位同学的部分代码，并进行调整和扩充完成了自动化测试）这个测试充分保证了除了`beq`指令之外指令的正确性。

测试电路





左图为我自己的`test1.0.circ`文件的电路，右图是`test_ans.circ`的电路图。

测试代码

```
import os
import re
import random
n=10
path=os.path.dirname(os.path.realpath(__file__))
os.chdir(path)
for j in range(n):
    with open("test.asm","w") as file:
        for i in range(10):
            x=random.randint(0,31)
            y=random.randint(0,31)
            # num=random.randint(0,10000)
            # file.write("lui $%d,%d\n"%(i,num))
            num=random.randint(0,10000)
            file.write("ori $%d,$%d,%d\n"%(x,y,num))
        for i in range(5):
            x=random.randint(0,31)
```

```
            y=random.randint(0,31)
            num=random.randint(0,100000)
            file.write("lw $%d,%d($%d)\n"%(x,num,y))
            file.write("sw $%d,%d($%d)\n"%(x,num,y))
        file.write("\n")
        for i in range(10):
            x=random.randint(0,31)
            y=random.randint(0,31)
            z=random.randint(0,31)
            #file.write("addu $%d,$%d,$%d\n"%(x,y,z))
            file.write("subu $%d,$%d,$%d\n"%(x,y,z))
    #生成并载入
    rom_name="rom"+str(j)+".txt"
    command="java -jar Mars4_5.jar test.asm nc mc CompactTextAtZero a dump .text
HexText "+rom_name
    os.system(command)
    content=open(rom_name).read()

    circmy=open("test1.0.circ").read()
    circmy=re.sub(r'addr/data: 5 32([\s\S]*)</a>',"addr/data: 5 32\n"+content+"
</a>",circmy)
    with open("test.circ","w") as file:
        file.write(circmy)

    circmy=open("test_ans.circ").read()
    circmy=re.sub(r'addr/data: 5 32([\s\S]*)</a>',"addr/data: 5 32\n"+content+"
</a>",circmy)
    with open("ans.circ","w") as file1:
        file1.write(circmy)

    os.system("java -jar logisim2.7.1.jar test.circ -tty table >my.txt")

    os.system("java -jar logisim2.7.1.jar ans.circ -tty table >ans1.txt")

    output = os.popen("fc my.txt ans1.txt").read()
    if output.__contains__("FC: no differences encountered"):
        print("#test point %d is successful!"%(j))
    else:
        print("#test point %d failed\n"%(j))
        print(output)

    os.system("fc my.txt ans1.txt")
```

测试结果

```
Administrator@MS-ENGERKGWZLHL MINGW64 /e/北航/大二上/2020年秋季学期课程学习/计组学习/CO_project_files/p3
$ python /e/北航/大二上/2020年秋季学期课程学习/计组学习/CO_project_files/p3/test.py
Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

Comparing files my.txt and ANS1.TXT
FC: no differences encountered

#test point 0 is successful!
#test point 1 is successful!
#test point 2 is successful!
#test point 3 is successful!
#test point 4 is successful!
#test point 5 is successful!
#test point 6 is successful!
#test point 7 is successful!
#test point 8 is successful!
#test point 9 is successful!
```

程序综合测试

进行了每条指令的测试之后。本文进行了整个CPU的综合测试。

程序1

MIPS源代码

```
ori  $t1, $zero, 0
ori  $t2, $t1, 2
ori  $t3, $zero, 1
ori  $a0, $zero, 5
ori  $a1, $zero, 3
addu $t1, $t1, $t3
addu $a0, $a0, $t3
subu $a1, $a1, $t3
beq  $t1, $t2, next
addu $t1, $t1, $t3
beq  $t1, $t2, next
next:subu $t1, $t1, $t3
lui  $t4, 0xfff
lui  $t6, 0x123
ori  $t2, $zero, 4
sw  $t6, 8($t2)
sw  $t4, 0x14($t2)
lw  $t5, 0x14($t2)
lw  $t7, 8($t2)
```

机器码

```
v2.0 raw
34090000
352a0002
340b0001
34040005
```

```
34050003
012b4821
008b2021
00ab2823
112a0002
012b4821
112a0000
012b4823
3c0c0fff
3c0e0123
340a0004
ad4e0008
ad4c0014
8d4d0014
8d4f0008
```

MIPS模拟结果

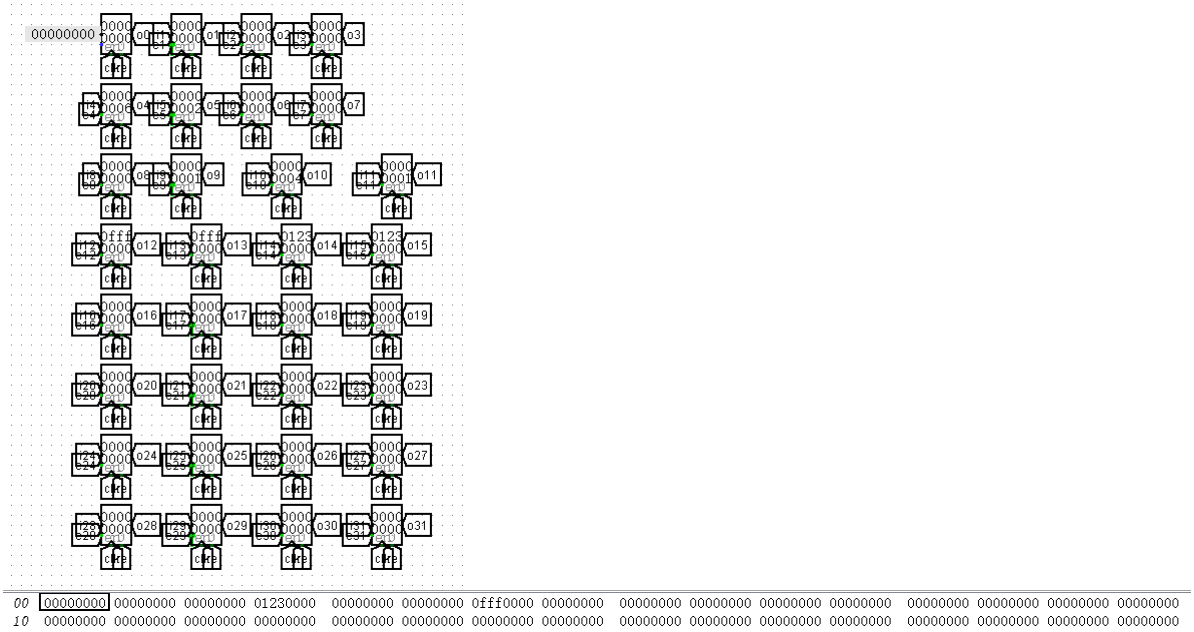
数据存储结果：

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000000	0x00000000	0x01230000	0x00000000	0x00000000	0x0fff0000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

寄存器数据：

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000006
$a1	5	0x00000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000001
$t2	10	0x00000004
$t3	11	0x00000001
$t4	12	0x0fff0000
$t5	13	0x0fff0000
$t6	14	0x01230000
$t7	15	0x01230000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000304c
hi		0x00000000
lo		0x00000000

我的CPU运行结果：



程序2

MIPS源代码

```
ori  $a0, $zero, 0x7cf
ori  $a1, $a0, 0x6f
lui  $a2, 0x3039
lui  $a3, 0xffff
ori  $a3, $a3, 0xffff
addu $s0, $a0, $a1
addu $s1, $a3, $a3
addu $s2, $a3, $s0
subu $s0, $a0, $s2
subu $s1, $a3, $a3
subu $s2, $a3, $a0
subu $s3, $s2, $s1
ori  $t0, $zero, 0
sw  $a0, ($t0)
nop
sw  $a1, 4($t0)
sw  $s0, 8($t0)
sw  $s1, 0xc($t0)
sw  $s2, 0x10($t0)
lw  $t7, ($t0)
lw  $t6, 0x14($t0)
sw  $t6, 0x18($t0)
lw  $t5, 0xc($t0)
ori  $t0, $t0, 1
ori  $t1, $t1, 1
ori  $t2, $t2, 2
beq  $t0, $t2, 0x28
beq  $t0, $t1, 0x74
lui  $t3, 0x457
addu $t0, $t0, $t7
```

机器码

```
v2.0 raw
340407cf
3485006f
3c063039
3c07ffff
34e7ffff
00858021
00e78821
00f09021
00928023
00e78823
00e49023
02519823
34080000
ad040000
00000000
ad050004
ad100008
ad11000c
ad120010
8d0f0000
8d0e0014
ad0e0018
8d0d000c
35080001
35290001
354a0002
110affef
11090001
3c0b0457
010f4021
```

与之前测试方法类似，对比MARS的运行结果和我的CPU运行结果，两者一致，因此本测试用例正确。

三、思考题

1.现在我们的模块中IM使用ROM， DM使用RAM， GRF使用Register，这种做法合理吗？请给出分析，若有改进意见也请一并给出。

答：ROM是只读存储器，适合存储固定不变的数据，而指令数据在单周期cpu运行过程有限个周期中不变的，因此选用ROM是合理的。RAM是读写存储器，DM要求存储器可以读和写，且速度要求不高，因此选取速度不高但是功能齐全的RAM比较合理。GRF需要经常对数据进行读写且速度要求比较高，因此选取Register比较合理。

2.事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

答：nop指令数据是0x00000000，除了PC=PC+4之外，没有进行任何其他操作，因此没有对电路中的逻辑真值运算产生任何影响，存在与否对电路无影响。

3.上文提到，MARS不能导出PC与DM起始地址均为0的机器码。实际上，可以通过为DM增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

答：假设DM存储大小是256MB，且数据地址从0x10000000-1fffffff，则将要存储的地址的最高四位和0x1进行比较，得到一个片选信号，相同则将这个数据存储到DM中，否则将数据存储到其他相应的位置。

4.除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索"形式验证（Formal Verification)"了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

答：所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。

对组合逻辑来说，不存在状态寄存器，其输出值仅仅依赖于当前的输入值。这时只要对每个输入值组合证明其输出的数据组合相同即可。

对时序逻辑而言，可以把它看成一个有限状态机。电路功能的等价可以用有限状态机的等价来判断。假定有两个状态机A和B，对他们的验证可以转化为以下的方法，当A和B有相同的接口，而且从相同的初始状态出发，两者对有效输入值序列产生相同的输出值序列，则可以说A和B等价。

形式验证的优点是：(1)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%。(2)形式验证技术是借用数学上的方法将待验证电路和功能描述或参考设计直接进行比较，不需要开发测试激励。(3)形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

缺点是：验证方法复杂抽象，难以准确把握。而且形式验证是数学逻辑分析，而不是电路分析，不能有效的验证电路的性能，如电路的时延和功耗等