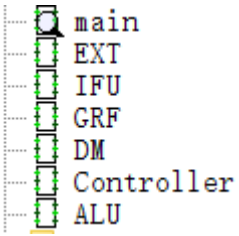
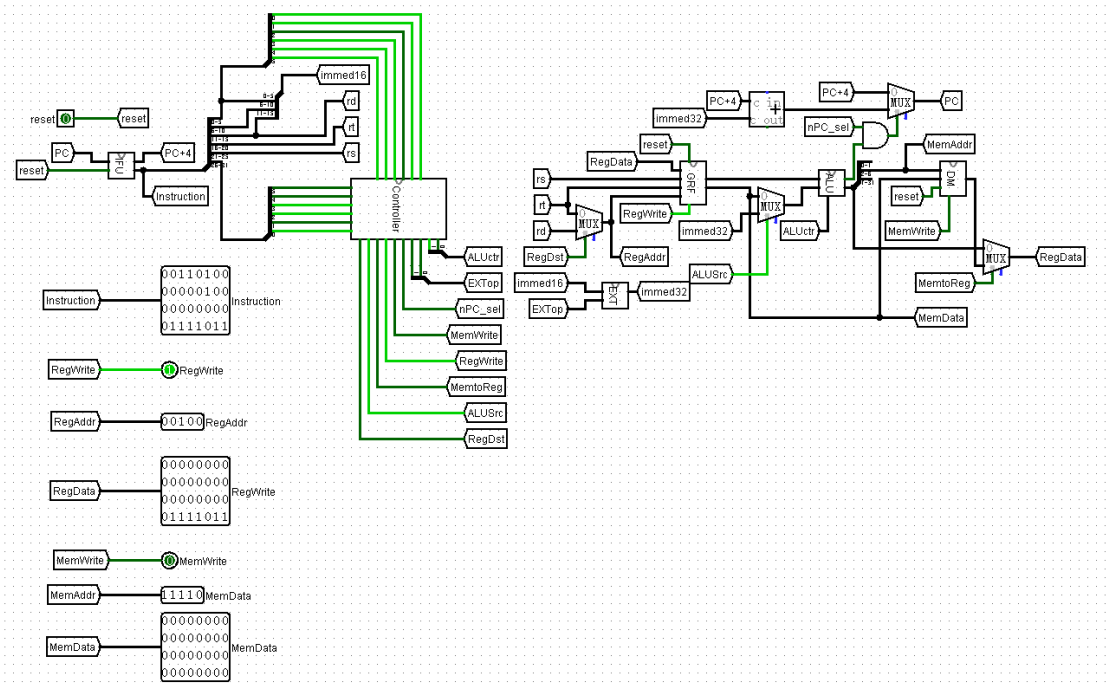
计算机组成原理实验报告

一、 CPU 设计文档

（一） 总体设计



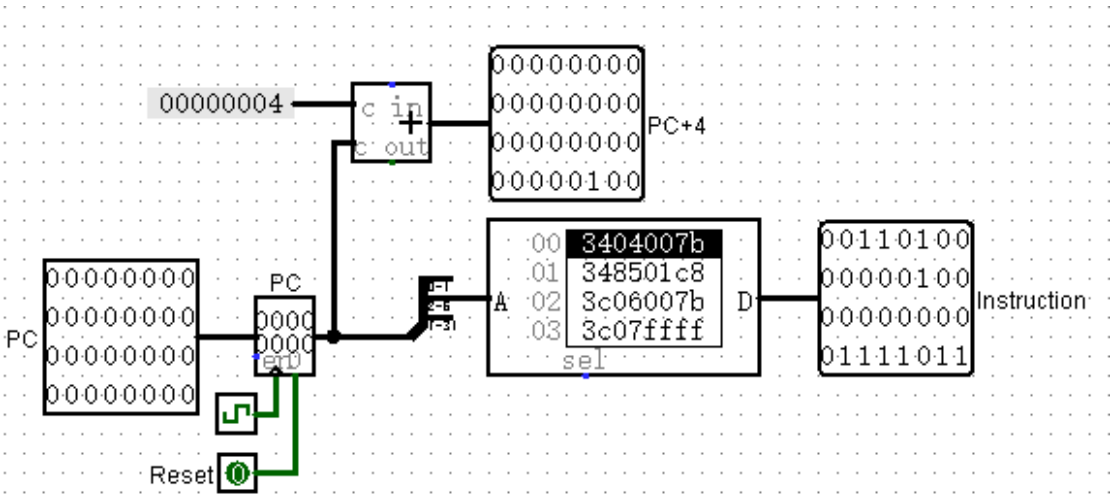
图表 1 模块设计



图表 2 总体设计

（二） 模块规格

1. IFU（取指令单元）



图表 3IFU

1）端口说明

表格 1 IFU 端口说明

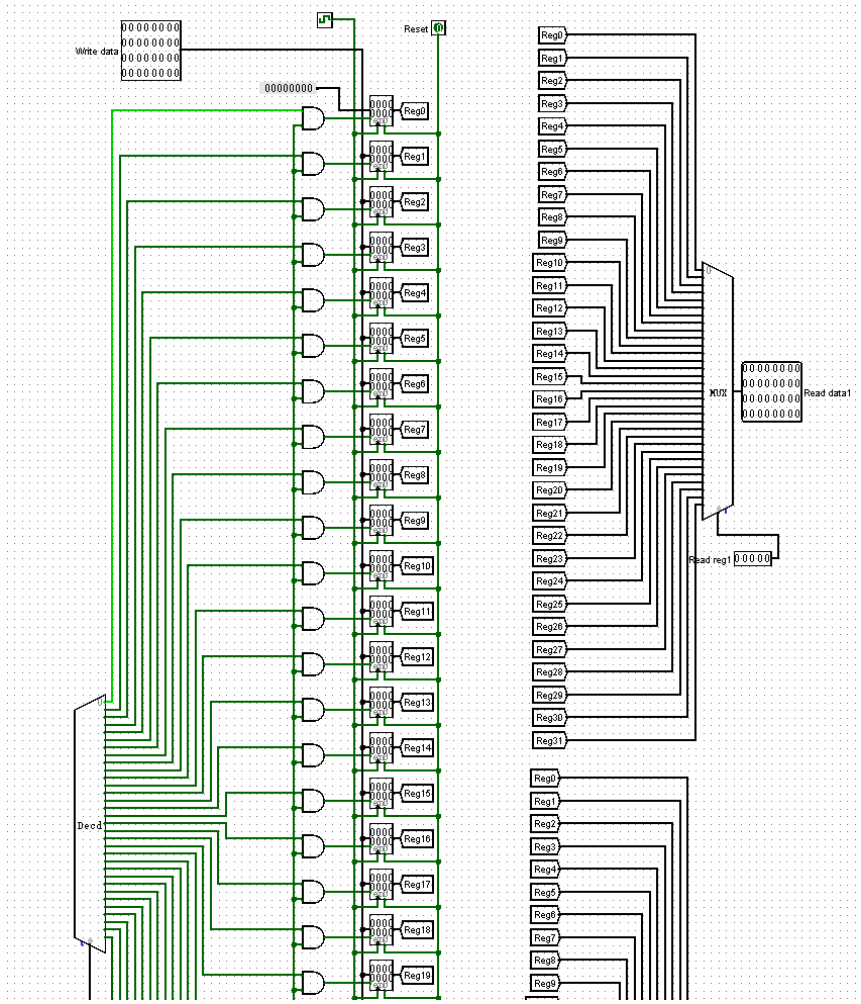
信号名	方向	描述
PC[31:0]	I	当前指令地址
Reset	I	复位信号 1：复位 0：无效
PC+4[31:0]	O	下一条指令地址
Instruction[31:0]	O	32 位 MIPS 指令

2）功能定义

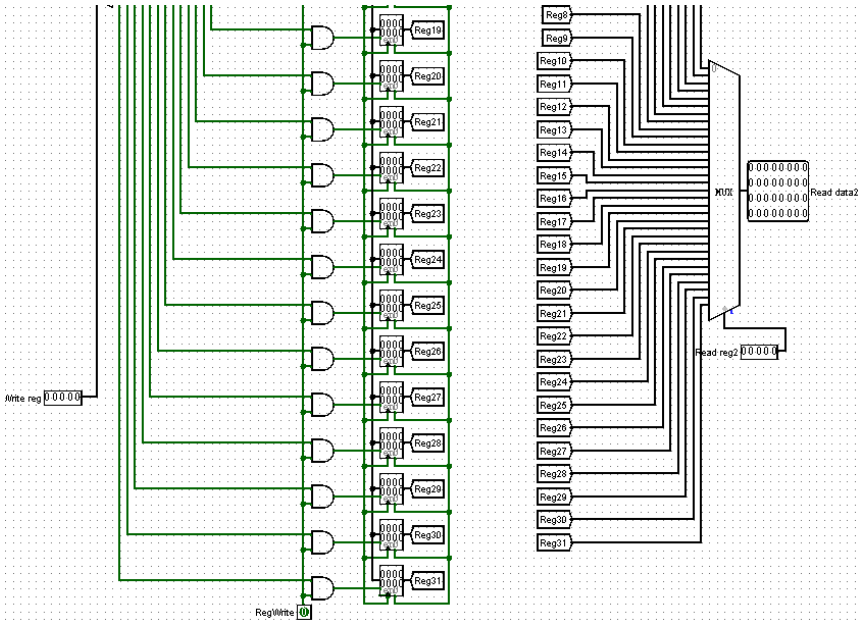
表格 2 IFU 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，PC 被置为 0x00000000
2	取指令	根据 PC 从 IM 中取出指令
3	计算下一条指令地址	PC←PC+4

2. GRF（通用寄存器组）



图表 4GRF



图表 5GRF

1）端口说明

表格 3GRF 端口说明

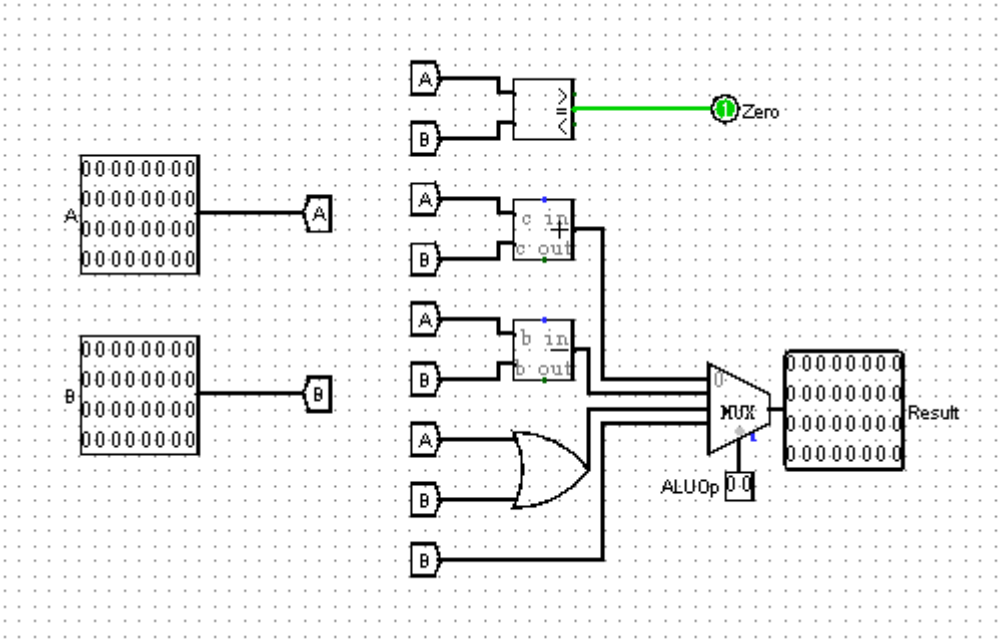
信号名	方向	描述
Write data[31:0]	I	写入的 32 位数据
Reset	I	复位信号 1：复位 0：无效
Read reg1[4:0]	I	读寄存器 1 编号
Read reg2[4:0]	I	读寄存器 2 编号
Write reg[4:0]	I	写寄存器编号
RegWrite	I	写控制信号 1：写入 0：无效
Read data1[31:0]	O	32 位寄存器 1 输出
Read data2[31:0]	O	32 位寄存器 2 输出

2）功能定义

表格 4GRF 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，32 个寄存器被置为 0x00000000
2	写寄存器	写寄存器控制信号有效时，把 32 位数据写入寄存器
3	读寄存器	根据输入的地址读出两个寄存器中的值

3. ALU（算术逻辑单元）



图表 6ALU

1）端口说明

表格 5ALU 端口说明

信号名	方向	描述
A[31:0]	I	32 位写入数据 1
B[31:0]	I	32 位写入数据 2
ALUOp[1:0]	I	控制信号 00：加运算 01：减运算 10：或运算 11：输出写入数据 2
Zero	O	相等信号 1：相等 0：不相等
Result[31:0]	O	32 位输出数据

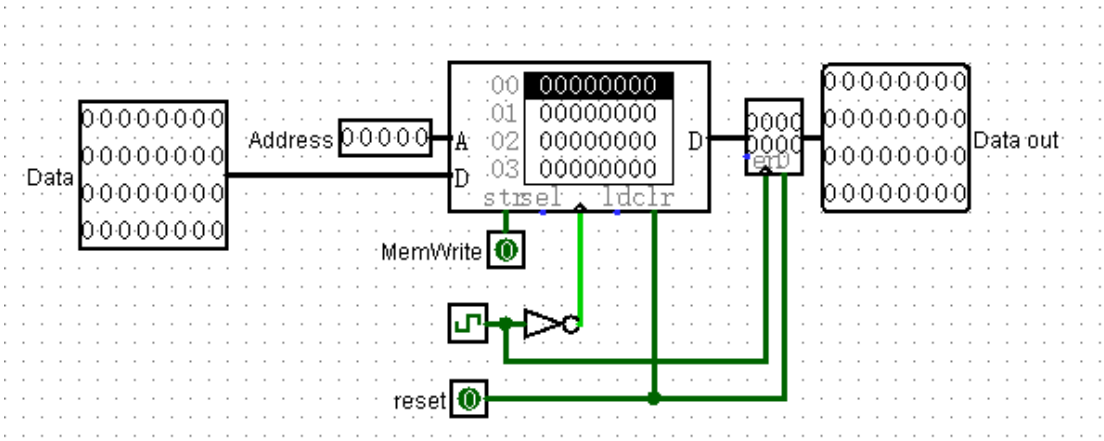
2）功能定义

表格 6ALU 功能定义

序号	功能名称	功能描述

1	加运算	A+B
2	减运算	A-B
3	或运算	A|B
4	输出写入数据 2	B
5	比较运算	A==B

4. DM（数据存储器）



图表 7DM

1）端口说明

表格 7DM 端口说明

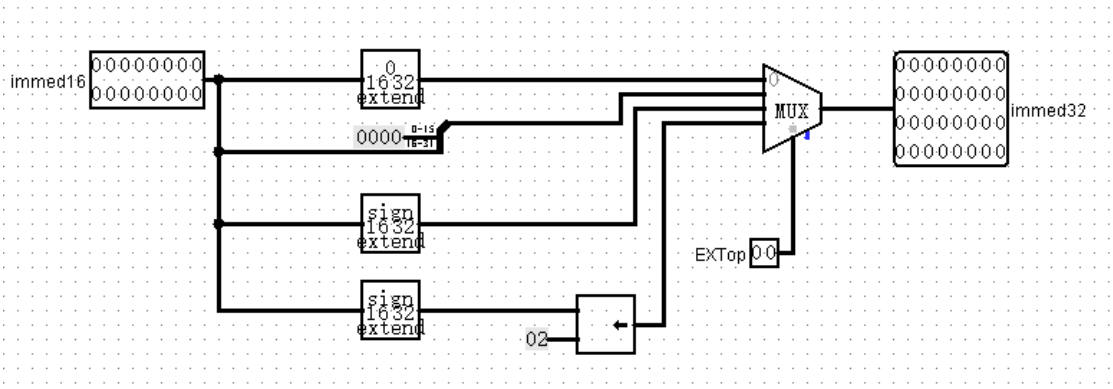
信号名	方向	描述
Address[4:0]	I	5 位写入内存地址
Data[31:0]	I	32 位写入数据
reset	I	复位信号 1：复位 0：无效
MemWrite	I	写内存控制信号 1：写入 0：无效
Dataout[31:0]	O	32 位输出数据

2）功能定义

表格 8DM 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，内存和读出内存的寄存器被置为 0x00000000
2	写内存	写内存控制信号有效时，根据输入的地址写入 32 位数据
3	读内存	根据输入的地址读出内存数据

5. EXT（位扩展器）



图表 8EXT

1）端口说明

表格 9EXT 端口说明

信号名	方向	描述
Immed16[15:0]	I	16 位写入立即数
EXTop[1:0]	I	扩展控制信号 00：无符号扩展 01：低 16 位补 0 10：有符号扩展 11：有符号扩展后逻辑左移两位

Immed32[31:0]	0	32 位输出立即数

2）功能定义

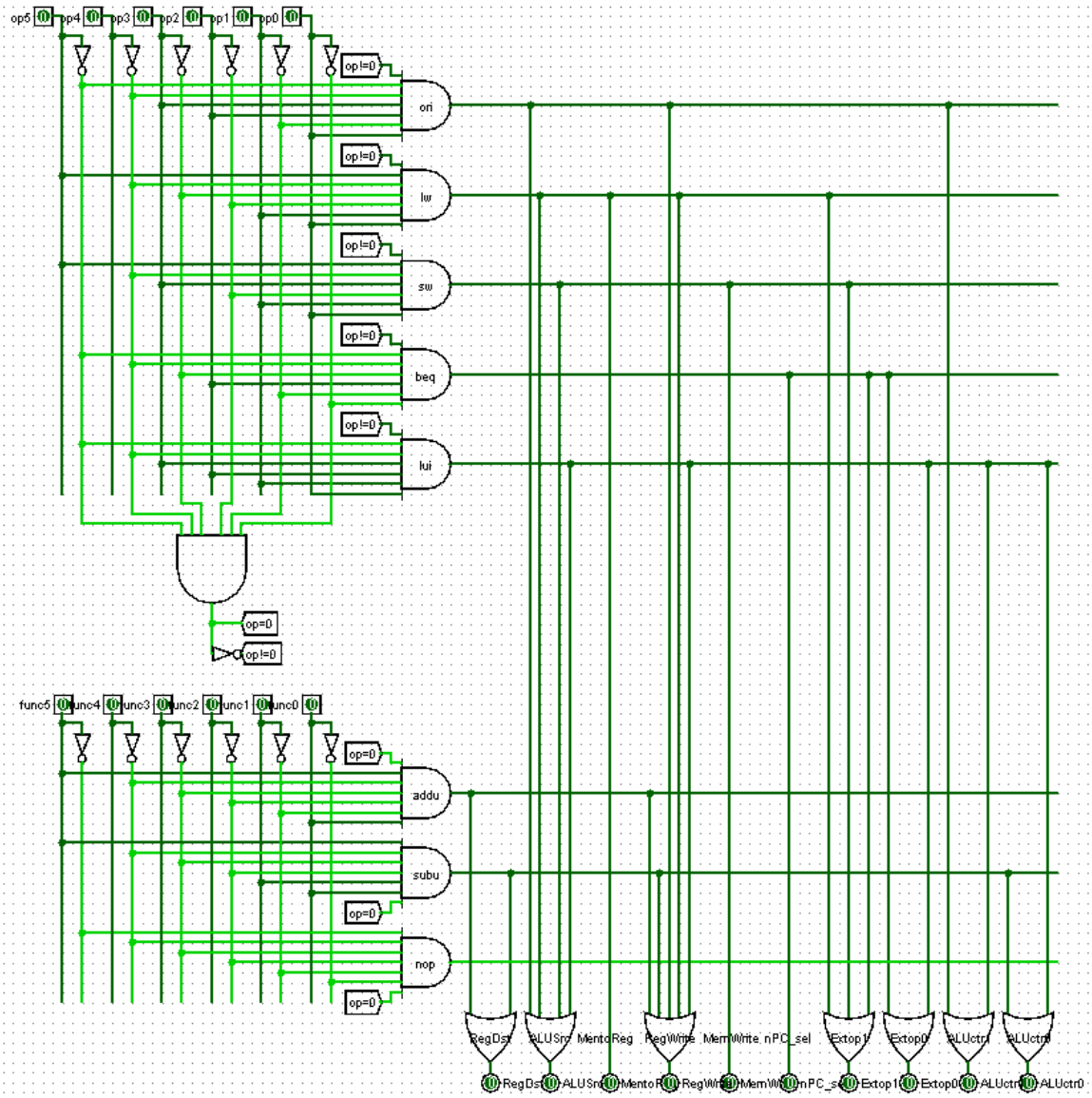
表格 10EXT 功能定义

序号	功能名称	功能描述
1	无符号扩展	高 16 位补 0
2	低 16 位补 0	低 16 位补 0
3	有符号扩展	Immed[15]为 1 时高 16 位补 1，为 0 时高 16 位补 0
4	有符号扩展后逻辑左移两位	Immed[15]为 1 时高 16 位补 1，为 0 时高 16 位补 0，再左移两位，溢出舍去，低 2 位补 0

6. Controller（控制器）

见第二章控制器设计。

（三） 控制器设计



图表 9Controller

1. 端口说明

表格 11Controller 端口说明

信号名	方向	描述
op[5:0]	I	6 位指令码 op
func[5:0]	I	6 位功能码 func
RegDst	0	GRF 写寄存器决定信号 1：rd 0：rt
ALUSrc	0	ALU 输入数据 B 决定信号 1：32 位立即数

		0：GRF 寄存器 2 输出值
MemtoReg	0	GRF 写入数据决定信号 1：DM 输出数据 0：ALU 输出数据
RegWrite	0	GEF 写寄存器信号 1：写寄存器 0：无效
MemWrite	0	写内存 DM 信号 1：写入内存 0：无效
nPC_sel	0	跳转信号 Zero 为 1 时： 1：跳转 0：无效 Zero 为 0 时： 无效
EXTop[1:0]	0	扩展控制信号 00：无符号扩展 01：低 16 位补 0 10：有符号扩展 11：有符号扩展后逻辑左移两位
ALUctr[1:0]	0	ALU 控制信号 00：加运算 01：减运算 10：或运算 11：输出写入数据 2

2. 真值表

表格 12Controller 真值表

func	100001	100011	N/A					
op	000000	000000	001101	100011	101011	000100	001111	000000
	addu	subu	ori	lw	sw	beq	lui	nop
RegDst	1	1	0	0	x	x	0	0
ALUSrc	0	0	1	1	1	0	1	0
MemtoReg	0	0	0	1	x	x	x	0
RegWrite	1	1	1	1	0	0	1	0
MemWrite	0	0	0	0	1	0	0	0
nPC_sel	0	0	0	0	0	1	0	0
EXTop	x	x	00	10	10	11	01	00
ALUctr	00	01	10	00	00	x	11	00

（四） 测试程序

（一） 测试程序

```
.data
.text
    #测试 ori 指令
    #第三个立即数是无符号扩展，不存在负数的情况
    ori $a0, $0, 123        #测试与 0 进行 or 运算
    ori $a1, $a0, 456       #测试两个非 0 数的 or 运算

    #测试 lui 指令
    lui $a2, 123            #测试，构造正数
    lui $a3, 0xffff         #测试，构造负数
```

```
#测试 addu 指令
#无符号相加，不存在负数情况
addu $s0, $a0, $a2    #测试正数相加

#测试 subu 指令
#无符号相减，不存在负数情况
subu $s1, $a2, $a0    #测试正数相减

#测试 sw 指令
ori $t0, $t0, 0x0000 #构造 0
sw $a0, 0($t0)        #测试 0111_1011 存入内存
sw $a1, 4($t0)        #测试 1_1111_1011 存入内存

#测试 lw 指令
lw $s0, 0($t0)        #测试 0111_1011 写入寄存器
lw $s1, 4($t0)        #测试 1_1111_1011 入寄存器

#测试 beq 指令
ori $a0, $0, 1        #构造 1
ori $a1, $0, 2        #构造 2
ori $a2, $0, 1        #构造 1
```

```
  beq $a0, $a1, loop1   #测试不跳转
  nop
  beq $a0, $a2, loop2   #测试跳转
  nop

loop1:ori $a0, $0, 0
loop2:ori $a1, $0, 0
```

（二） 机器码

```
3404007b
348501c8
3c06007b
3c07ffff
00868021
00c48823
35080000
ad040000
ad050004
8d100000
8d110004
34040001
34050002
```

```
34060001

10850003

00000000

10860002

00000000

34040000

34050000
```

（三） 期望输出

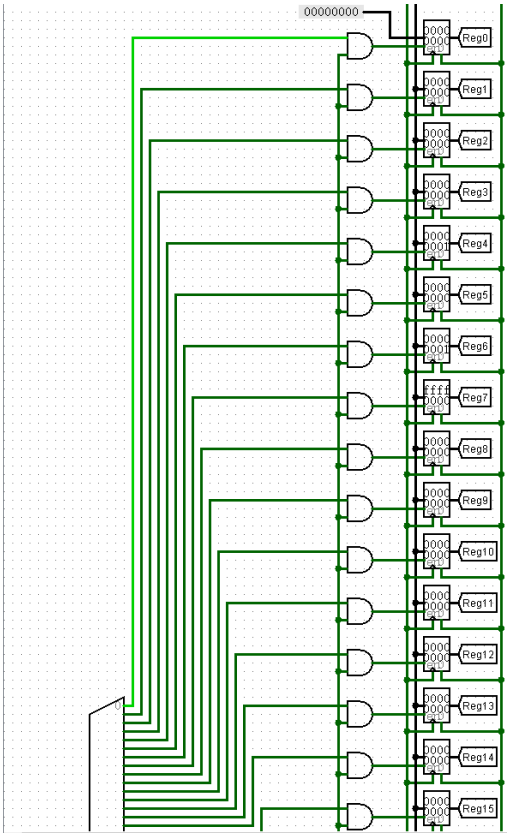
```
$a0 0x00000001

$a2 0x00000001

$a3 0xffff0000

$s0 0x0000007b

$s1 0x000001fb
```

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000001
$a1	5	0x00000000
$a2	6	0x00000001
$a3	7	0xffff0000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x0000007b
$s1	17	0x000001fb
$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		0x00003050
hi		0x00000000
lo		0x00000000

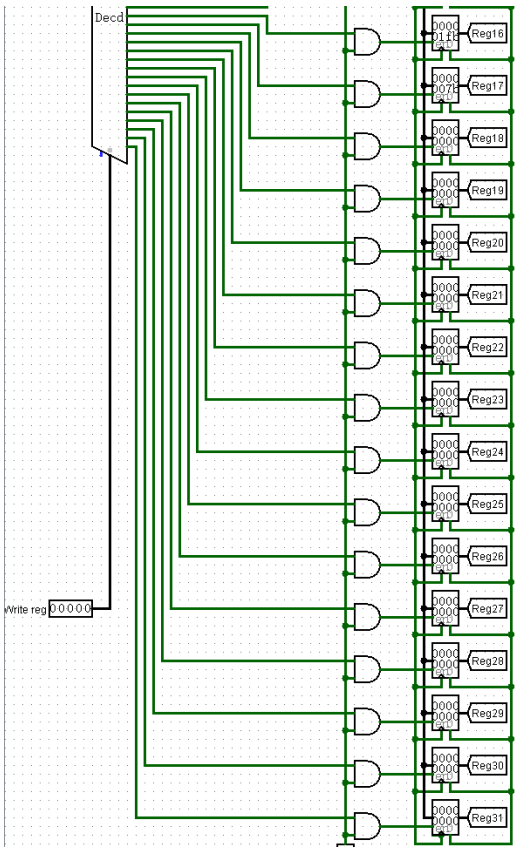
图表 10MARS 输出

（四） Logisim 输出

GRF：

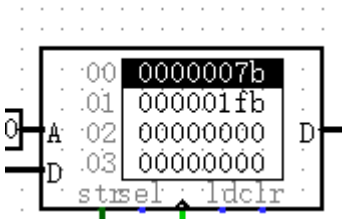


图表 11logisim 输出 1



图表 12logisim 输出 2

DM：



图表 13logisim 输出 3

（五） 结论

期望输出与实际输出相同。

二、 思考题

（一） 若 PC（程序计数器）位数为 30 位，试分析其与 32 位 PC 的优劣。

30 位 PC 能寻址 256M 个字的位置，32 位则能寻 1GB 字的位置，因此 32 位寻址范围更广。然而 30 位 PC 使用 PC+1，取[4：0]位，32 位 PC 使用 PC+4，取[6:2]位，实际上是相同的。但因 beq 等指令中，跳转时需要相加的立即数为 32 位，采用 30 位 PC 非常不便，因此采用 32 位 PC 更方便。

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

合理。IM 存储机器码，不应该改变，只需读出即可，因此使用 ROM；DM 为内存，其数据包含写入和读取，应使用 RAM；GRF 需要快速读写，以匹配 CPU 速度，因此用寄存器。

（三） 结合上文给出的样例真值表，给出 RegDst， ALUSrc，MemtoReg，RegWrite， nPC_Sel， ExtOp 与 op 和 func 有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

RegDst

$$= \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ \overline{func[1]}\ func[0]$$

$$+ \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ func[1]\ func[0]$$

$$ALUSrc = \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$
$$+ op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ op[5]\ \overline{op[4]}\ op[3]\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ op[1]\ op[0]$$

$$MemtoReg = op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$

RegWrite

$$= \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ \overline{func[1]}\ func[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ func[1]\ func[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ \overline{op[1]}\ op[0]$$
$$+ op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ op[1]\ op[0]$$

$$MemWrite = op[5]\ \overline{op[4]}\ op[3]\ \overline{op[2]}\ op[1]\ op[0]$$

$$nPC\_sel = \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

$$EXTop[1] = op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ op[5]\ \overline{op[4]}\ op[3]\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

$$EXTop[0] = \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

（四） 充分利用真值表中的 X 可以将以上控制信号化简为最简

单的表达式， 请给出化简后的形式。

RegDst

$$= \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ func[0]$$

$$ALUSrc = \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ \overline{op[0]} + op[5]\ \overline{op[4]}\ \overline{op[2]}\ op[1]\ op[0]$$

$$MemtoReg = op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$

RegWrite

$$= \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ \overline{op[1]}\ \overline{op[0]}\ func[5]\ \overline{func[4]}\ \overline{func[3]}\ \overline{func[2]}\ func[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ op[3]\ op[2]\ op[0] + op[5]\ \overline{op[4]}\ \overline{op[3]}\ \overline{op[2]}\ op[1]\ op[0]$$

$$MemWrite = op[5]\ \overline{op[4]}\ op[3]\ \overline{op[2]}\ op[1]\ op[0]$$

$$nPC\_sel = \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

$$EXTop[1] = op[5]\ \overline{op[4]}\ \overline{op[2]}\ op[1]\ op[0]$$
$$+ \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

$$EXTop[0] = \overline{op[5]}\ \overline{op[4]}\ \overline{op[3]}\ op[2]\ \overline{op[1]}\ \overline{op[0]}$$

（五） 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。

nop 指令机器码为 0x00000000，不进行任何有效行为，因此不会影响控制信号真值表。

（六） 前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM 片选信号，就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数

据偏移。

data base address[31:0]=0x00003000, 因此，data base address[12]=data base address[13]=1，其余位都为零，将第 12 或 13 位作为 DM 中 RAM 的 str 和 ld 信号之一控制 ld，MemWrite 为 1 时控制 str，只有当第 12 或 13 位为 1 时才有可能对 DM 进行读写。

（七） 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，简要阐述相比与测试，形式验证的优劣。

形式验证的优点：不用考虑输入具体数据，克服了不能穷举所有情况的问题，耗时较短，减小设计周期。

缺点：过于抽象，对抽象思维的要求高。