Project3 Logisim 完成单周期处理器开发实验报告

一. 整体结构

1.处理器为 32 位处理器。

2.处理器应支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}。

3.nop 机器码为 0x00000000， 即空指令，不进行任何有效行为（修改寄存器等）

4.addu,subu 可以不支持溢出。

5.处理器为单周期设计。

6.需要采用模块化和层次化设计。顶层有效的驱动信号要求包括且仅包括：reset (clk 请使用内置时钟模块).

二. 模块规格

1. IFU

模块接口

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
Zero	I	ALU 计算结果是否为 0 的标志信号 1：计算结果为 0 0：计算结果非 0
Branch	I	当前指令是否为 beq 指令 1：是 beq 0:不是 beq
Instr	O	32 位 MIPS 指令 做输出
Imm32	I	beq 指令的 sign_extend(offset ||0^2) 32 位立即数

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 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	如果当前指令不是 beq 指令，则 PC=PC+4 如果当前指令是 beq 指令，且 Zero=1，则 PC=PC+4+sign_extend(offset ||0^2)

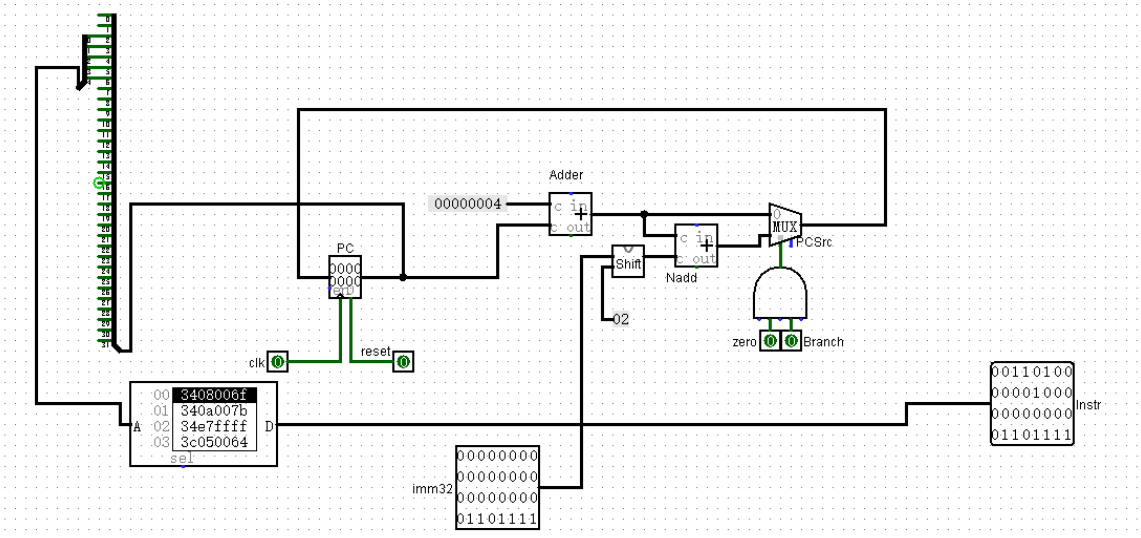
IM

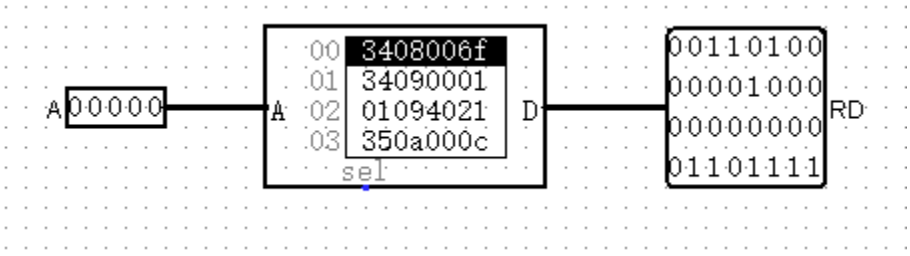
模块接口

信号名	方向	功能描述
A[4:0]	I	5 位地址
RD[31:0]	O	32 位指令

功能定义

序号	功能名称	功能描述
1	取指令	根据 PC 从 IM 中取出指令





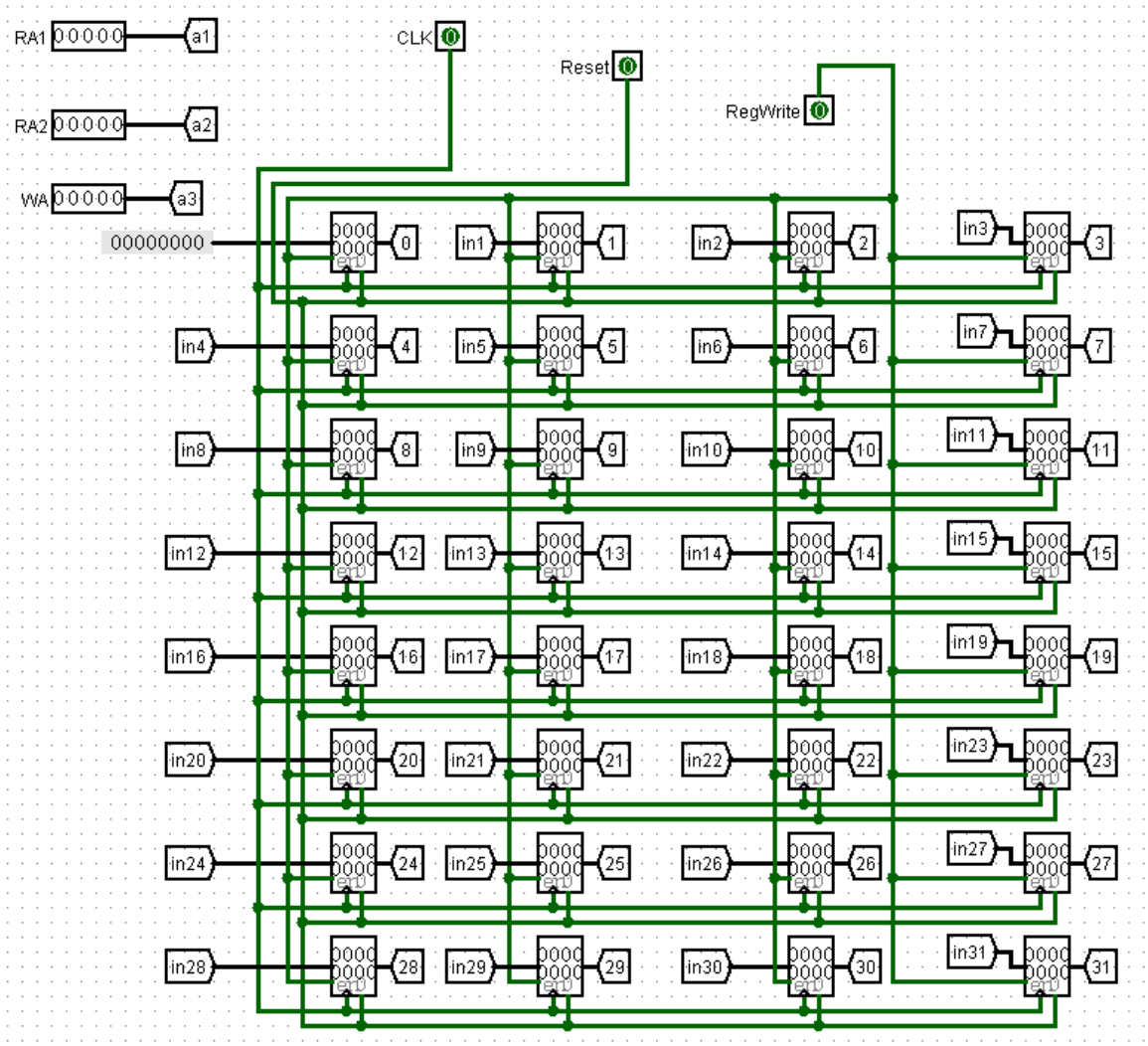
2. GRF

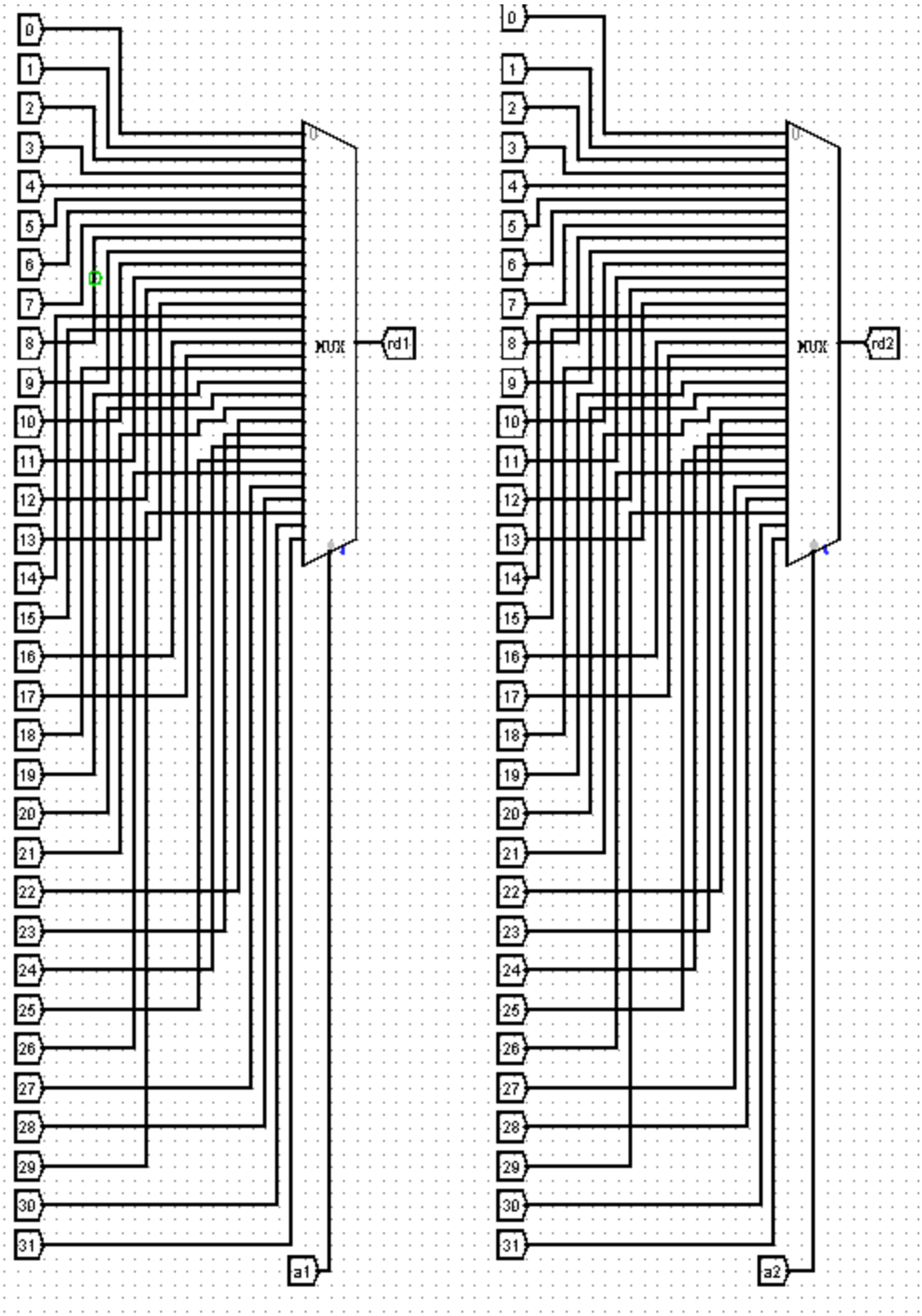
模块接口

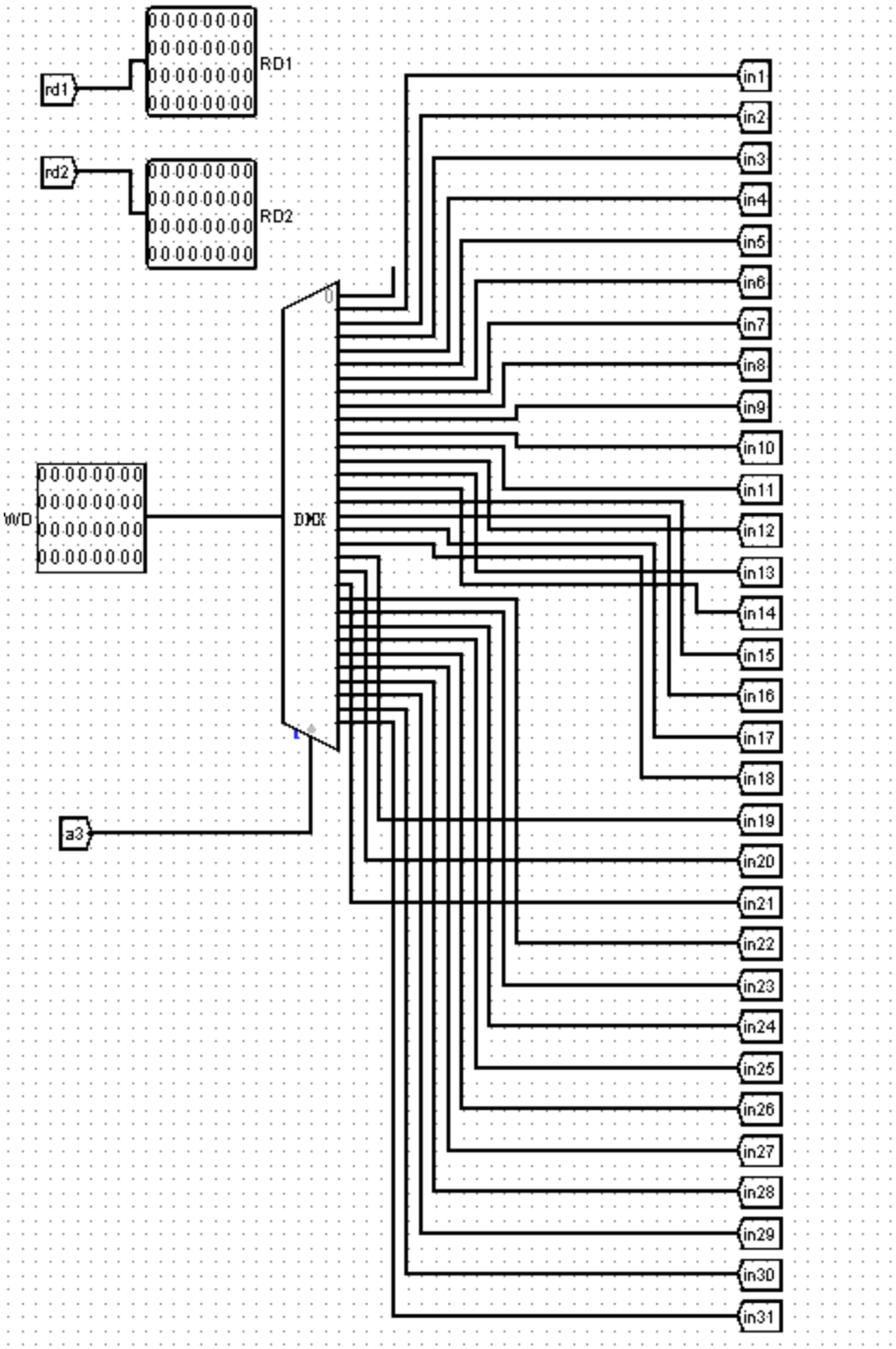
信号名	方向	功能描述
WD[31:0]	I	写入数据的输入
RA1[4:0]	I	读寄存器地址 1
RA2[4:0]	I	读寄存器地址 2
WA[4:0]	I	写寄存器地址
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
RegWrite	I	是否可以写入控制信号(随时都可以读出) 1：可以写 0：不可以写
RD1[31:0]	O	32 位数据输出 1
RD2[31:0]	O	32 位数据输出 2

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器被设置为 0x00000000
2	读寄存器	根据输入的寄存器地址读出 32 位数据
3	写寄存器	根据输入的地址，把输入的数据写进所选的寄存器


RA1
00000
a1
RA2
00000
a2
WA
00000
a3
00000000
CLK
Reset
RegWrite
in1
in2
in3
in4
in5
in6
in7
in8
in9
in10
in11
in12
in13
in14
in15
in16
in17
in18
in19
in20
in21
in22
in23
in24
in25
in26
in27
in28
in29
in30
in31
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
MUX
rd1
a1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
MUX
rd2
a2


rd1
RD1
rd2
RD2
WD
a3
0
1
in1
in2
in3
in4
in5
in6
in7
in8
in9
in10
in11
in12
in13
in14
in15
in16
in17
in18
in19
in20
in21
in22
in23
in24
in25
in26
in27
in28
in29
in30
in31

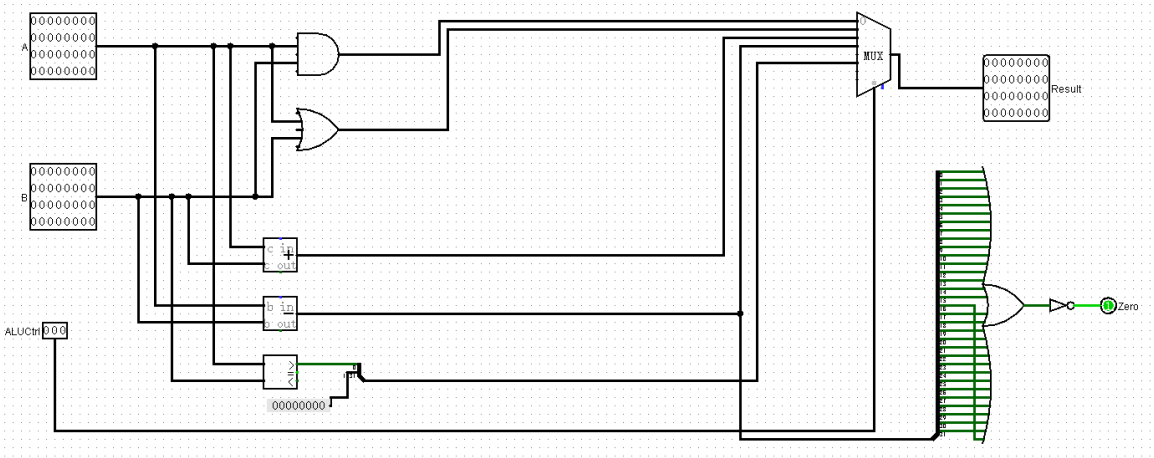
3. ALU

模块接口

信号名	方向	功能描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
ALUCtrl[2:0]	I	控制信号 000：与 001：或 010：加 011：减 101：比较大小
Result[31:0]	O	32 位数据输出
Zero	O	A,B 是否相等的标志信号 1：相等 0：不相等

功能定义

序号	功能名称	功能描述
1	与	A&B
2	或	A|B
3	加	A+B
4	减	A-B
5	判零	A=?B
6	比较大小	A>B?A<B



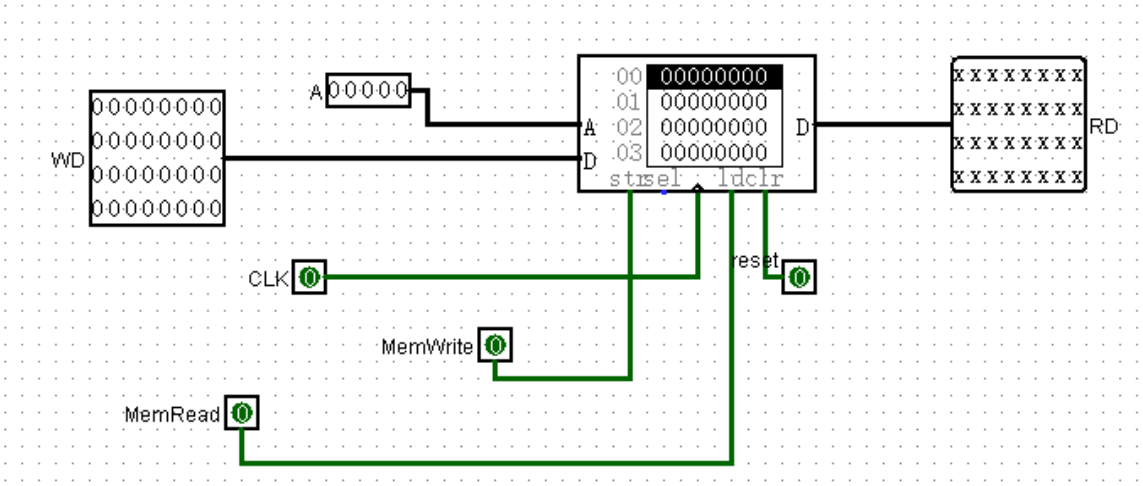
4. DM

模块接口

信号名	方向	功能描述
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
MemWrite	I	读写控制信号 1：写操作
MemRead	I	读写控制信号 1：读操作
A[4:0]	I	操作寄存器地址
WD[31:0]	I	输入（写入内存）的 32 位数据
RD[31:0]	O	32 位数据输出

功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有数据被设置为 0x00000000
2	读	根据输入的寄存器地址读出数据
3	写	根据输入的地址，把输入的数据写入



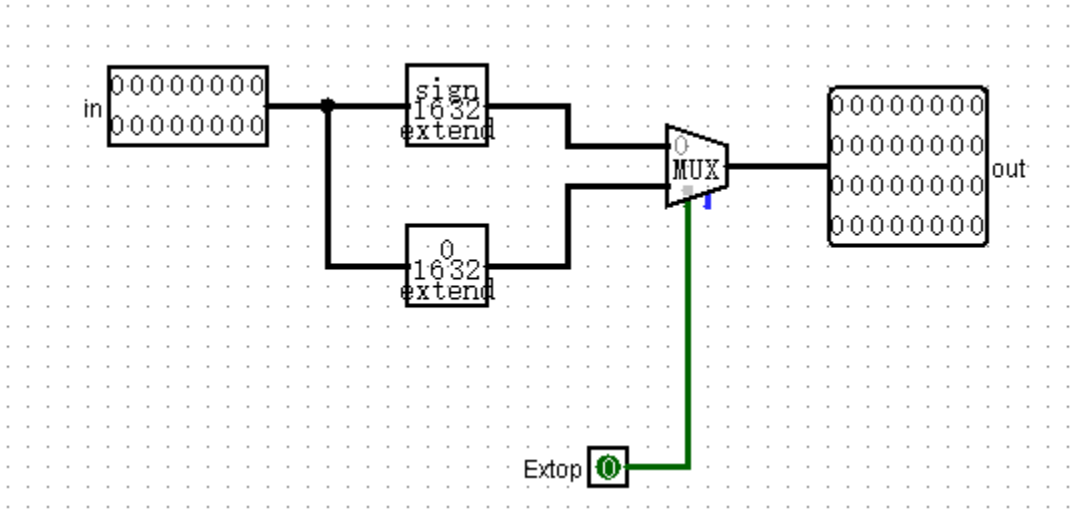
5. EXT

模块接口

信号名	方向	功能描述
In[15:0]	I	16 位数据输入
Out[32:0]	O	32 位数据输出
ExtOp	I	控制信号 0：高位符号扩展 1：高位 0 扩展

功能定义

序号	功能名称	功能描述
1	高位符号扩展	高 16 位补符号位
2	高位 0 扩展	高 16 位补 0



6. Controller

模块接口

信号名	方向	功能描述
Op[5:0]	I	6 位 opcode 段
RegDst	O	写地址控制 选择 RT,RD
ALUSrc	O	ALU 第二操作数选择控制
RegWrite	O	GRF 写入控制
MemRead	O	DM 读信号
MemWrite	O	DM 写信号
MemToReg[1:0]	O	GRF 写入数据的选择信号
ExtOp	O	高位扩展方式选择信号
Branch	O	判断是否为 beq 指令的信号 是则为 1
ALUOp[2:0]	O	传递给 ALUControl 与 func[5:0]共同确定 ALUctrl[3:0]的信号

功能定义

序号	功能名称	功能描述
1	产生控制信号	产生控制信号

7.ALUControl

模块接口

信号名	方向	功能描述
ALUOp[2:0]	I	由 op[5:0]产生的控制信号
Func[5:0]	I	6 位 funcode
ALUCtrl[3:0]	O	ALU 选择信号

功能定义

序号	功能名称	功能描述
1	产生 ALU 控制信号	产生 ALU 控制信号，控制 ALU 进行不同的运算

三．控制器设计思路

数据通路如下

指令	Adder		PC	IM.A	GRF				ALU		DM		EXT	Nadd		Shift
	A	B			RA1	RA2	WA	WD	ALU	B	A	WD		A	B	
R 型	PC	4	Adder	PC	Rs	Rt	Rd	ALU	RF.RD1	RF.RD2						
lw	PC	4	Adder	PC	Rs		Rt	DM.RD	RF.RD1	sign_ext	ALU		imm16			
sw	PC	4	Adder	PC	Rs	Rt			RF.RD1	sign_ext	ALU	RF.RD2	imm16			
beq	PC	4	Adder /Nadd	PC	Rs	Rt			RF.RD1	RF.RD2			imm16	Adder	Shift	Sign_ext
ori	PC	4	Adder	PC	Rs		Rt	ALU	RF.RD1	zero_ext			imm16			
lui	PC	4	Adder	PC			Rt	imm+016								
nop	PC	4	Adder	PC												

由此可见需要以下几个 MUX 多路选择器

1.beq 指令 PC 有两种选择 PC=Adder 输出或者 Nadd 的输出 选择信号为 Branch

2.GRF 的 WA 端选择 Rd,Rt 需要一个 MUX，控制信号 RegDst

3.GRF 的 WD 输入端，有三种选择：RF.RD2，ALU 的输出，lui 指令直接对 imm16 后边补 16 位 0，需要 2 选 4MUX,选择信号 MemToReg[1:0]

4.两种扩展方式的选择（符号扩展，0 扩展）选择信号 EXTOp

5.ALU 的 B 端两种选择，RF.RD2 或 EXT 的输出，选择信号 ALUSrc

除了上述 Branch, ALUSrc, EXTOp, MemToReg[1:0], RegDst, 还有三个读写控制信号，RegWrite 是 GRF 写入信号，MemRead, MemWrite 是 DM 读写信号，所以控制器 Controller 需要设计这 8 个控制信号。除此之外，设定 ALUOp[2:0]用以作为 ALUController 的输入，与 func[5:0]一起，决定 ALU 的控制信号 ALUCtrl[2:0],即决定 ALU 的运算方式（加减与或）。

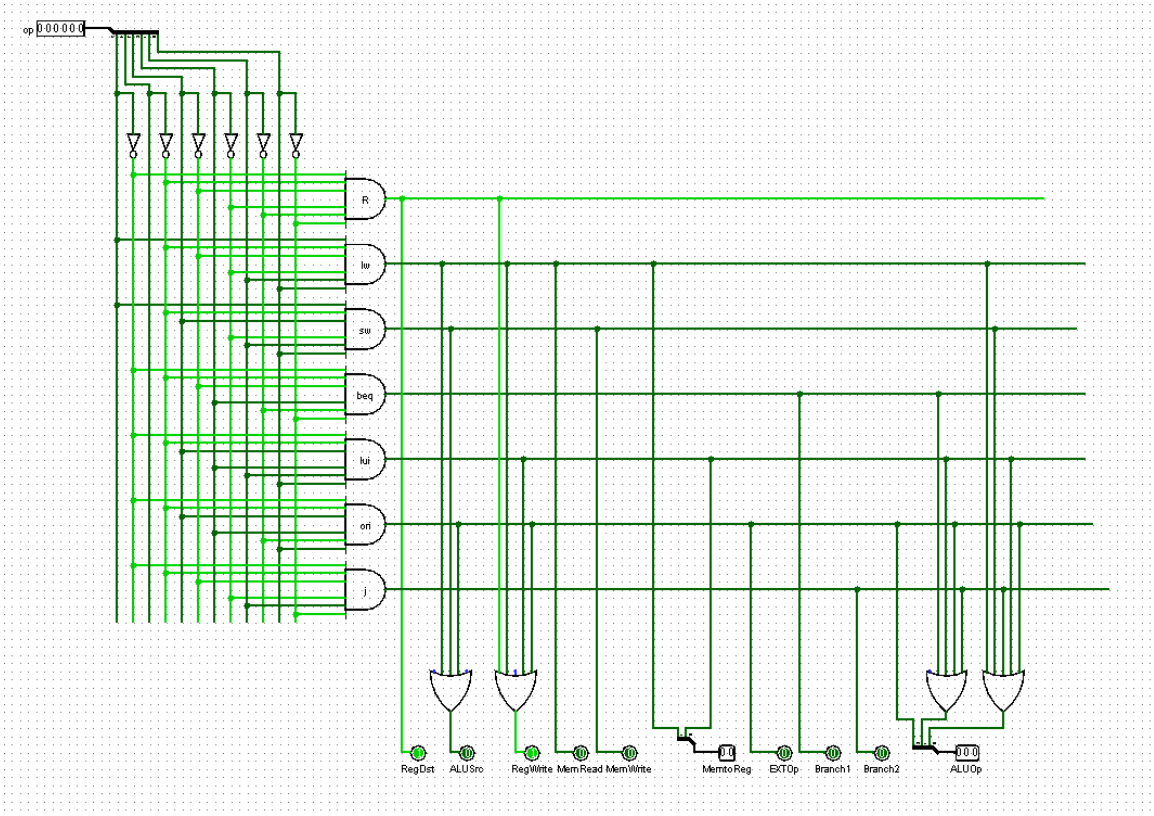
信号名	方向	功能描述
Op[5:0]	I	6 位 opcode 段
RegDst	O	写地址控制 选择 RT,RD
ALUSrc	O	ALU 第二操作数选择控制
RegWrite	O	GRF 写入控制
MemRead	O	DM 读信号
MemWrite	O	DM 写信号
MemToReg[1:0]	O	GRF 写入数据的选择信号
ExtOp	O	高位扩展方式选择信号
Branch	O	判断是否为 beq 指令的信号 是则为 1
ALUOp[2:0]	O	传递给 ALUControl 与 func[5:0]共同确定 ALUctrl[3:0]的信号

可以绘制如下表格

name	R	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	x0	x0	x1	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



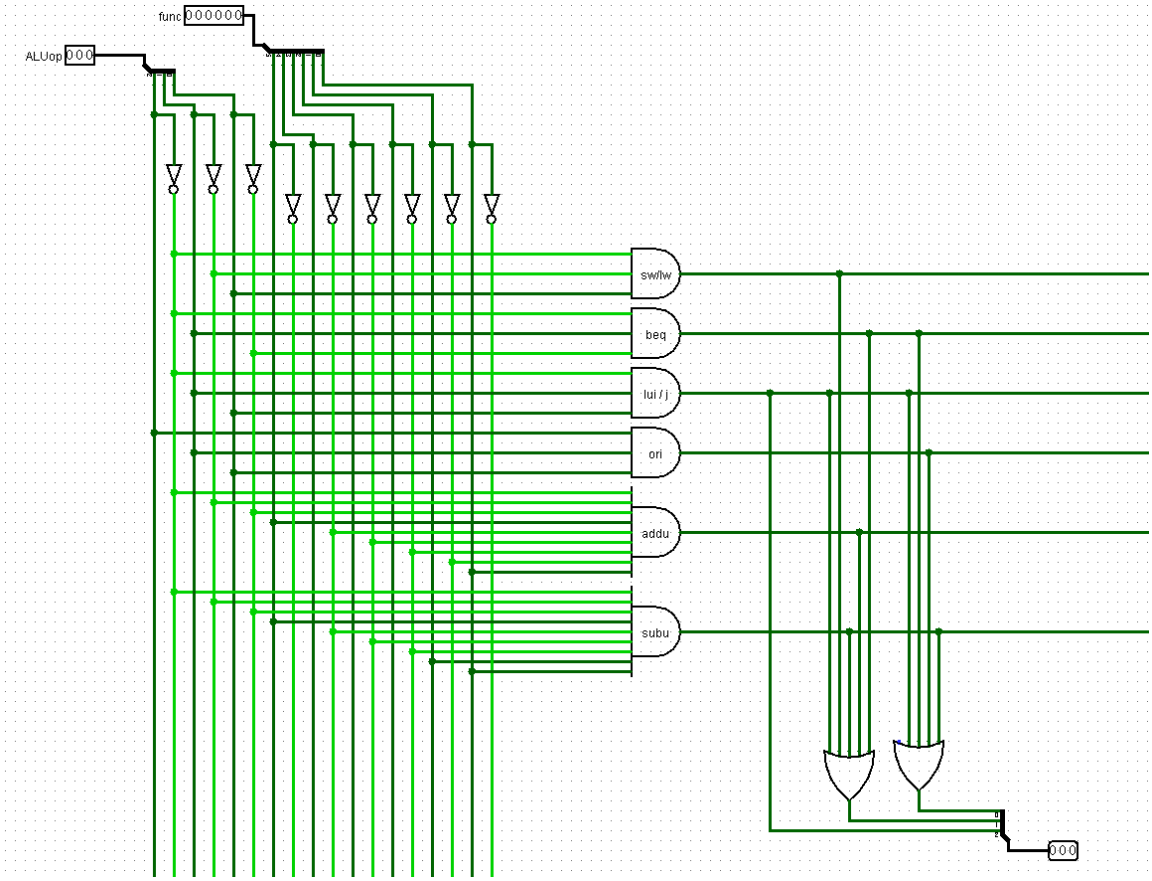
ALUController 的设计

R 类型的 ALUCtrl 需要关心 Func[5:0],而其他类型只关心由 Controller 产生的 ALUOp[2:0],根据实际意义确定 ALU 的运算类型，画表入下

指令	Func[5:0]	ALUOp[2:0]	类型	ALUCtrl[2:0]

lw	xxxxxx	001	加	010
sw	xxxxxx	001	加	010
beq	xxxxxx	010	减	011
lui	xxxxxx	011	x	111
ori	xxxxxx	111	或	001
addu	100001	000	加	010
subu	100011	000	减	011

与或门阵列



四．测试程序

ori $a0,$0,1999　　#ori 测试程序要实现：$0 寄存器中的内容与立即数 0x000007cf 进行或运算，储存在$a0 寄存器中

ori $a1,$a0,111 #ori 测试程序要实现：$a0 寄存器中的内容与立即数 0x0000006f 进行或运算，储存在$a1 寄存器中

lui $a2,12345 #lui 测试程序要实现：立即数 0x00003039 加载至 $a2 寄存器的高位

lui $a3,0xffff #lui 测试程序要实现：立即数 0x0000ffff 加载至 $a3 寄存器的高位

ori $a3,$a3,0xffff #ori 测试程序要实现：$a3 寄存器中的内容与立即数 0x0000ffff 进行或运算，储存在$a3 寄存器中

addu $s0,$a0,$a1 #addu 测试程序要实现：a0 寄存器中的值加上 a1 后存到 s0 寄存器中

addu $s1,$a3,$a3 #addu 测试程序要实现：a3 寄存器中的值加上 a3 后存到 s1 寄存器中

addu $s2,$a3,$s0 #addu 测试程序要实现：a3 寄存器中的值加上 s0 后存到 s2 寄存器中

subu $s0,$a0,$s2 #subu 测试程序要实现：a0 寄存器中的值减去 s2 寄存器中的值后存到 s0 寄存器中

subu $s1,$a3,$a3 #subu 测试程序要实现：a3 寄存器中的值减去 a3 寄存器中的值后存到 s1 寄存器中

eee:

subu $s2,$a3,$a0 #subu 测试程序要实现：a3 寄存器中的值减去 a0 寄存器中的值后存到 s2 寄存器中

subu $s3,$s2,$s1 #subu 测试程序要实现：s2 寄存器中的值减去 s1 寄存器中的值后存到 s3 寄存器中

ori $t0,$0,0x0000 #ori 测试程序要实现：$0 寄存器中的内容与立即数 0x00000000 进行或运算，储存在$t0 寄存器中

sw $a0,0($t0) #sw 测试程序要实现：把 a0 寄存器中值,存储到 t0 寄存器的值再加上偏移量 0， 所指向的 RAM 中

```
nop

sw $a1,4($t0)        #sw 测试程序要实现：把 a1 寄存器中值,存储到 t0 寄存器的值再加上偏移量 4， 所指向的 RAM 中

sw $s0,8($t0)        #sw 测试程序要实现：把 s0 寄存器中值,存储到 t0 寄存器的值再加上偏移量 8， 所指向的 RAM 中

sw $s1,12($t0)       #sw 测试程序要实现：把 s1 寄存器中值,存储到 t0 寄存器的值再加上偏移量 12， 所指向的 RAM 中

sw $s2,16($t0)       #sw 测试程序要实现：把 s2 寄存器中值,存储到 t0 寄存器的值再加上偏移量 16， 所指向的 RAM 中

lw $t7,0($t0)        #lw 测试程序要实现：把 t0 寄存器的值加上偏移量 0 当作地址读取存储器中的值存入 t7

lw $t6,20($t0)       #lw 测试程序要实现：把 t0 寄存器的值加上偏移量 20 当作地址读取存储器中的值存入 t6

sw $t6,24($t0)       #sw 测试程序要实现：把 t6 寄存器中值,存储到 t0 寄存器的值再加上偏移量 24， 所指向的 RAM 中

lw $t5,12($t0)       #lw 测试程序要实现：把 t0 寄存器的值加上偏移量 12 当作地址读取存储器中的值存入 t5

ori $t0,$t0,1        #ori 测试程序要实现： $t0 寄存器中的内容与立即数 0x00000001 进行或运算，储存在$t0 寄存器中

ori $t1,$t1,1        #ori 测试程序要实现： $t1 寄存器中的内容与立即数 0x00000001 进行或运算，储存在$t1 寄存器中

ori $t2,$t2,2        #ori 测试程序要实现： $t2 寄存器中的内容与立即数 0x00000002 进行或运算，储存在$t2 寄存器中

beq $t0,$t2,eee      #beq 测试程序要实现：判断 t0 的值和 t2 的值是否相等，相等转 eee

beq $t0,$t1,end      #beq 测试程序要实现：判断 t0 的值和 t1 的值是否相等，相等转 end
```

```
lui $t3,1111      #lui 测试程序要实现：立即数 0x00000457 加载至 $t3 寄存器的高位

end:

addu $t0,$t0,$t7    #addu 测试程序要实现：t0 寄存器中的值加上 t0 后存到 t0 寄存器中
```

机器码

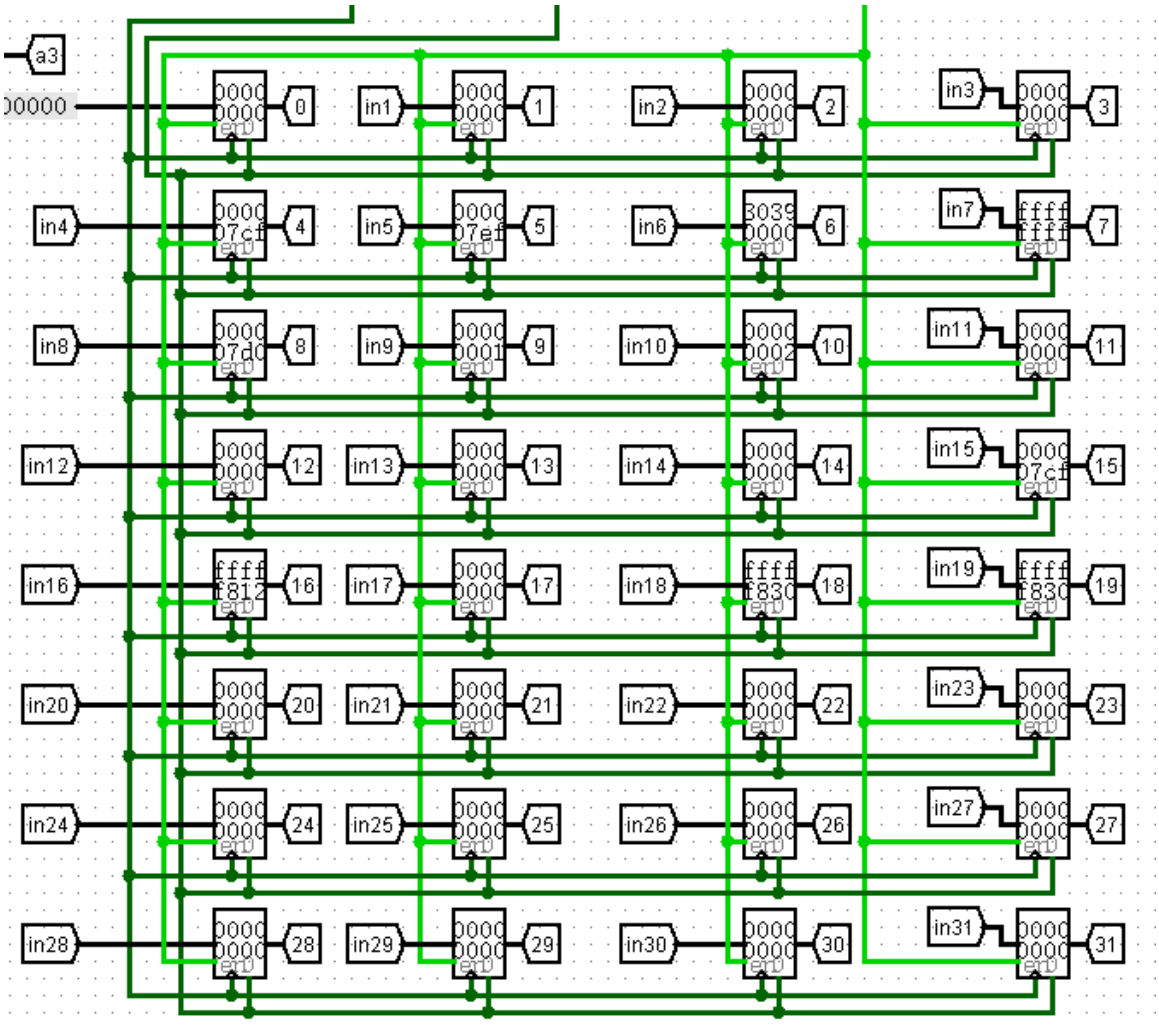
```
v2.0 raw

340407cf

3485006f

3c063039

3c07ffff

34e7ffff

00858021

00e78821

00f09021

00928023

00e78823

00e49023

02519823

34080000

ad040000

00000000

ad050004
```

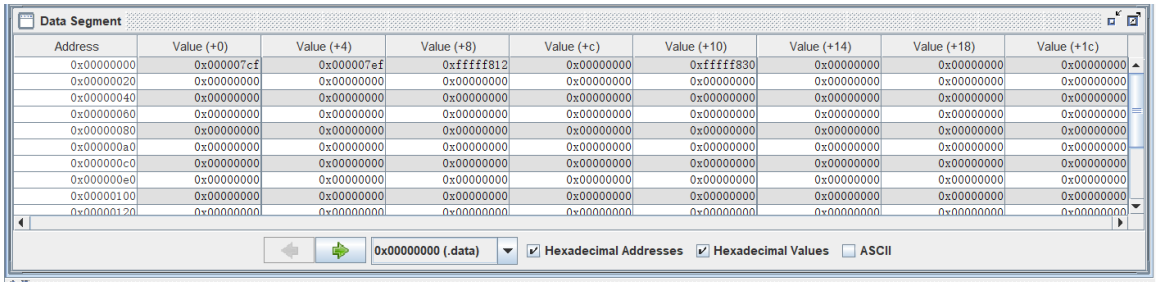
```
ad100008

ad11000c

ad120010

8d0f0000

8d0e0014

ad0e0018

8d0d000c

35080001

35290001

354a0002

110affef

11090001

3c0b0457

010f4021
```

MARS 模拟结果如下

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x000007cf
$a1	5	0x000007ef
$a2	6	0x30390000
$a3	7	0xffffffff
$t0	8	0x000007d0
$t1	9	0x00000001
$t2	10	0x00000002
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x000007cf
$s0	16	0xfffff812
$s1	17	0x00000000
$s2	18	0xfffff830
$s3	19	0xfffff830
$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		0x00003078
hi		0x00000000
lo		0x00000000

GRF





DM

```
00  000007cf 000007ef fffff812 00000000  fffff830 00000000 00000000 00000000
08  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
10  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
18  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
```

五．思考题

L0. T2

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

32 位 PC 可以直接进行 PC+4，不用扩展,30 位 PC 需要扩展

2.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，ROM 是只读存储器，作为 IM 指令存储器来存储指令，只负责读出指令，而 DM 用来想内存中存储数据，可读可写，不需要速度很快，但需要足够的存储空间，RAM 正适合，而 GRF 是寄存器堆，速度要快，选择寄存器，寄存器是这三种原件中速度最快的了，所以很合适。

L0. T3

1.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。）

Func f5,f4,f3,f2,f1

Op o5,o4,o3,o2,o1

与&& 或|| 非!

RegDst=!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0

ALUSrc=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)||(o5&&!o4&&o3&&!o2&&o1&&o0)

MemtoReg=o5&&!o4&&!o3&&!o2&&o1&&o0

RegWrite=(!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0)||(!o5

&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)

nPC_Sel=!o5&&!o4&&!o3&&o2&&!o1&&!o0

ExtOp=o5&&!o4&&!o2&&o1&&o0

2.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。

将 X 按照方便化简的原则当成 0 或 1

RegDst=!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0

ALUSrc=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)||(o5&&!o4&&o3&&!o2&&o1&&o0)=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o2&&o1&&o0)

MemtoReg=o5&&!o4&&!o3&&!o2&&o1&&o0

RegWrite=(!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0)||(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)

nPC_Sel=!o5&&!o4&&!o3&&o2&&!o1&&!o0

ExtOp=o5&&!o4&&!o2&&o1&&o0

3. 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 是 0X00000000,只是做了 PC=PC+4，别的什么都没有修改，没对逻辑电路电路中的元件进行任何操作，存在与否对电路没有影响。

L0. T4

1.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。

假设 DM 有 256MB 容量,并且映射在 0x3000_0000-0x3FFF_FFFF 区间,那么只需要把高 4 位地址与 0x3 进行比较,结果就是 DM 的片选信号。之前的 DM 存满后就从 0x3000_0000~0x3FFF_FFFF 存储数据。这次的 DM 最多存到 0x0000_00fc,所以不需要片选信号。

2.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。

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。

1.组合逻辑电路的逻辑验证

(1)转换为单一抽象模型比较。通过对单一表示的结构进行比较，得出其功能等价的结论。在最坏的情况下，布尔函数为正，表示随输入个数指数增加，其过大的内存需求限制了一般布尔函数的验证能力。

(2)利用测试输入向量进行验证。探寻使两个电路具有不同输出的输入测试向量，若存在这样的测试向量，则电路在功能上等价。在最坏情况下，这种方法需要穷举所有可能的输入测试矢量，运行时间又成为一个主要问题。

2．时序逻辑电路的验证

对一个时序电路而言，可以把它看成一个有限状态机(FSM，finite-state machine)。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，要对它们进行比较。直观的说，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：

1. 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。

2. 形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了测试验证的不足。对指定描述的所有可能的情况进行验证，覆盖率达到了 100%。

3. 形式验证可以进行从系统级到门级的验证，验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，缩短设计周期。

形式验证的缺点：

形式验证到目前为止仍然不能有效的验证电路的性能，如电路的时延和功耗等