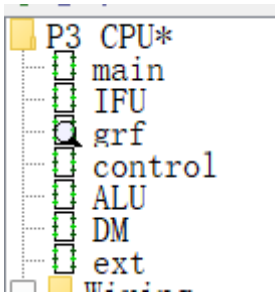
P3-logisim开发单周期处理器设计报告

一，整体设计



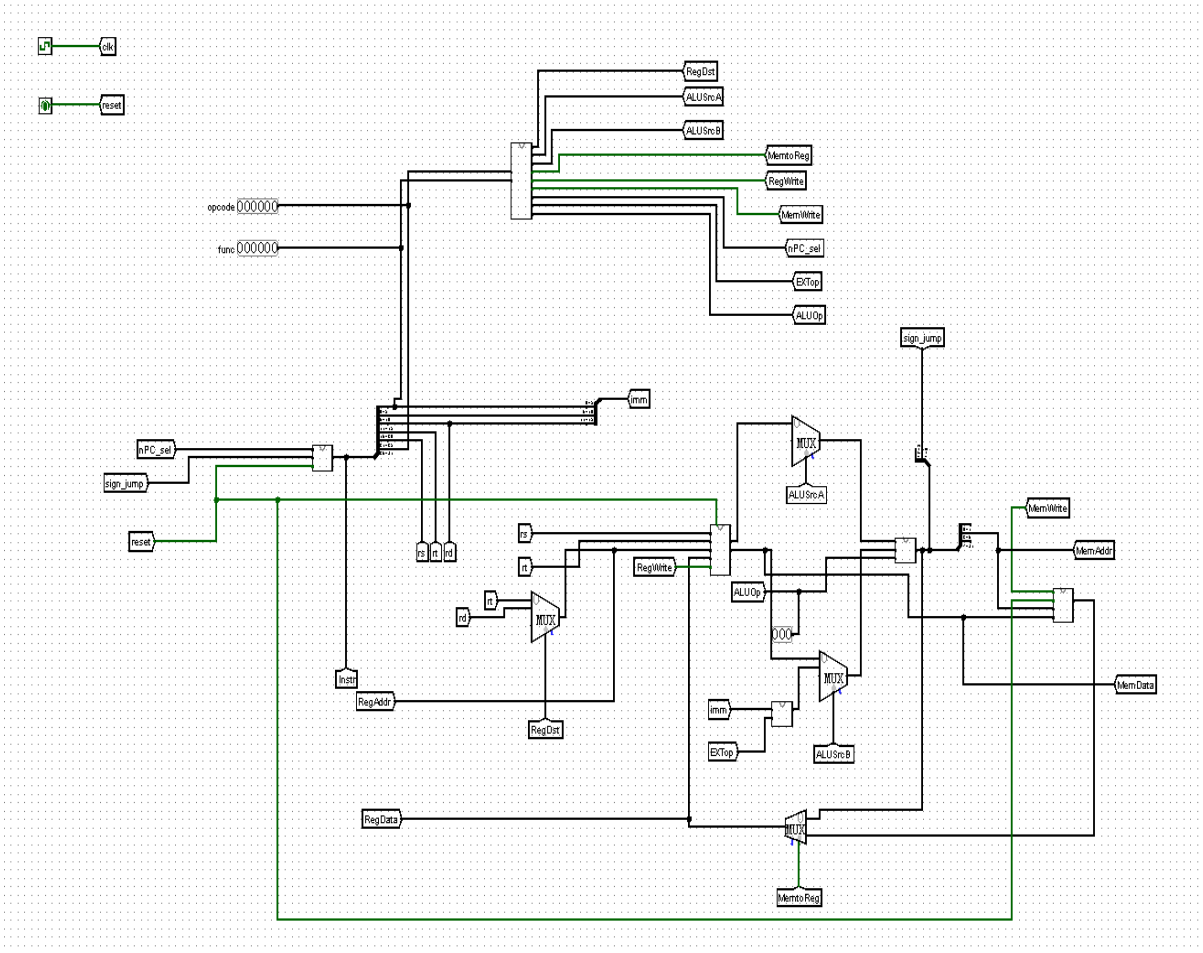


图 1-cpu 设计总览

二，模块定义

（1）IFU

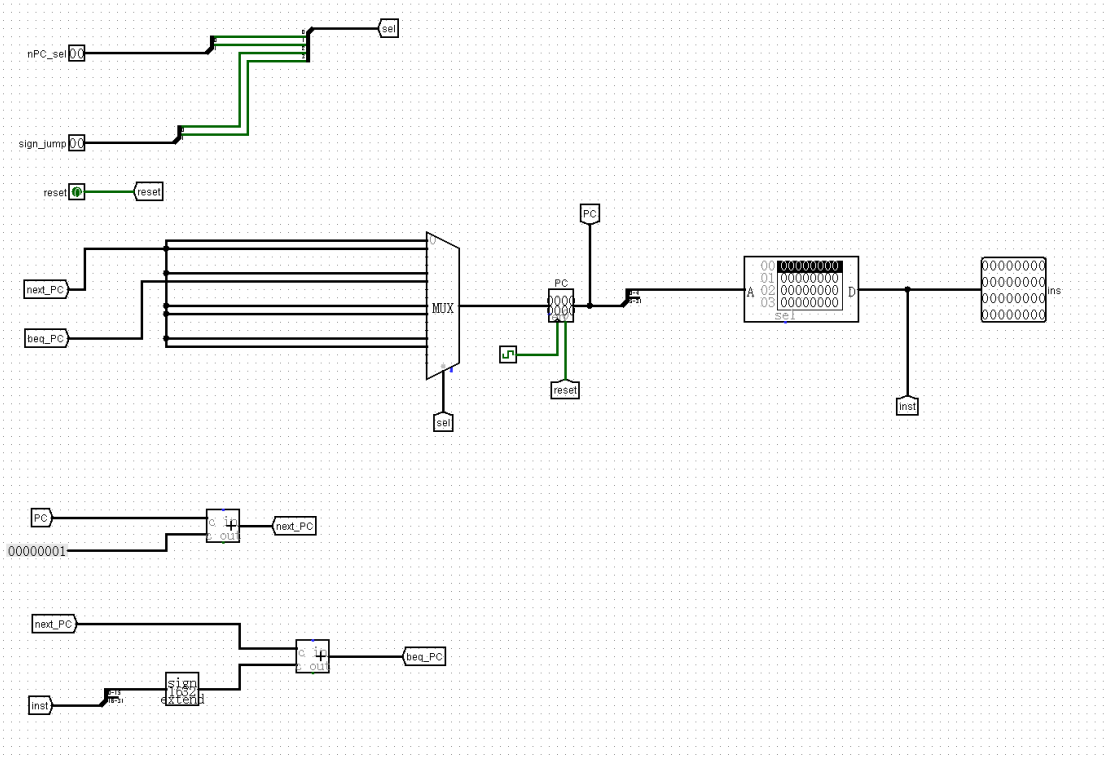


图 2-IFU 设计

表 1-IFU 模块说明

端口名称	方向	端口定义
nPC_sel[1:0]	I	判断当前指令是否为 beq 当 nPC_seq 为 00 时，当前指令不为 beq 当 nPC_seq 为 01 时，当前指令为 beq
sign_jump[1:0]	I	判断当前指令执行时，是否满足[rs] == [rt] 当 zero 等于 00 时，不满足该条件 当 zero 等于 01 时，满足该条件
reset	I	复位信号 当 reset 为 0 时，电路正常运行 当 reset 为 1 时，复位寄存器 PC
clk	I	时钟信号
Ins[31:0]	O	32 为 Mips 指令

表 2-IFU 功能说明

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 寄存器被设置为 0x00000000
2	取指令	当时钟上升沿到来时，从 IM 中取出指令并输出
3	计算下一条指令地址	根据 sel 选择下一条指令 若 sel 为 0101，则选择 beq_PC 若 sel 为 0000,0100,1000,1100,0001,1001,1101,则选择 next_PC

（2）GRF

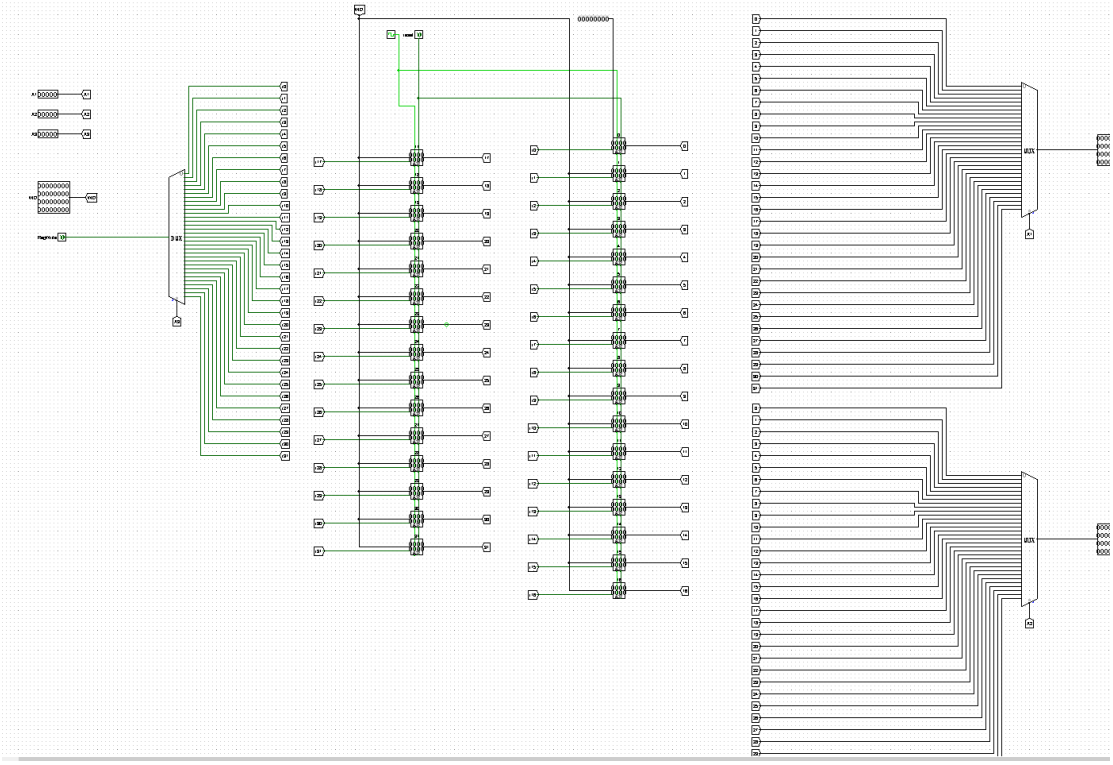


图 3-GRF 设计

表 3-GRF 模块说明

端口名称	方向	端口定义
A1	I	读寄存器地址 1
A2	I	读寄存器地址 2
A3	I	写寄存器地址
WD	I	写入数据
RegWrite	I	读写控制信号 0：读操作 1：写操作

reset	I	复位信号
RD1	O	32 位输出 1
RD2	O	32 位输出 2

表 4-GRF 功能说明

序号	功能名称	功能描述
1	读	根据 A1,A2 输出相应寄存器中的数据
2	写	根据 A3 向相应寄存器中写入数据
3	复位	当复位信号有效时清空所有寄存器数据

（3）ALU

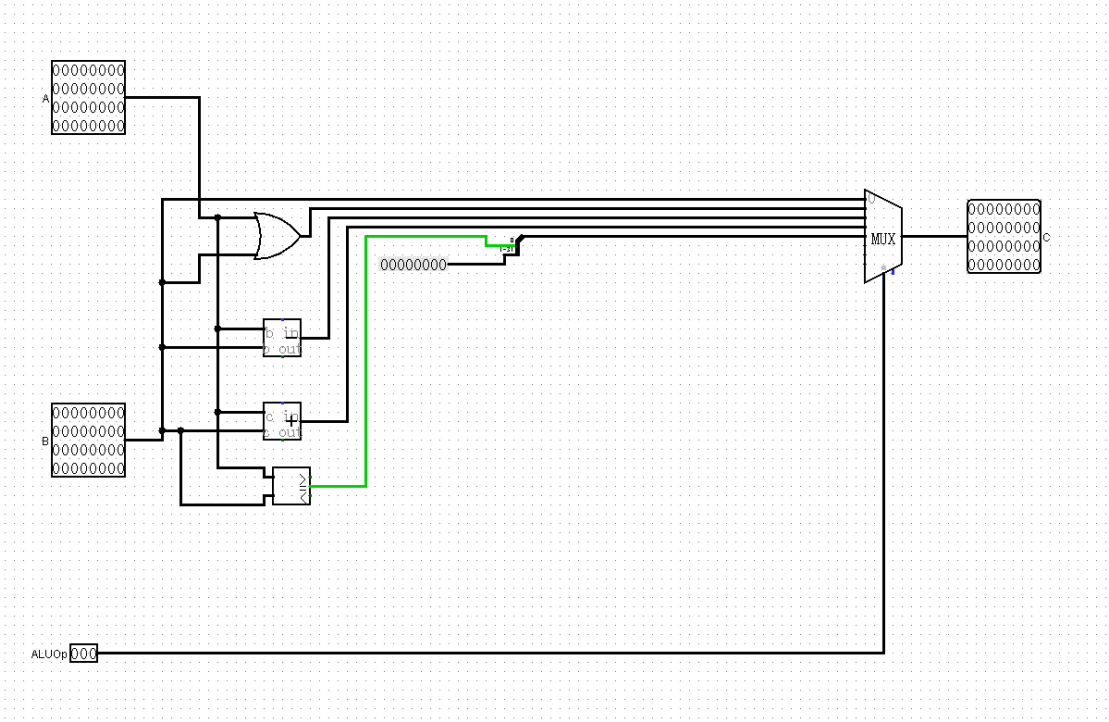


图 4-ALU 设计

表 5-ALU 模块说明

端口名称	方向	端口定义
A[31:0]	I	第一个 32 位运算数据
B[31:0]	I	第二个 32 位运算数据

ALUOp[2:0]	I	控制信号： 000：将第二个运算数直接作为输出结果 001：或运算 010：减法运算 011：加法运算 100：判断是否相等
C[31:0]	O	32 位计算结果

表 6-ALU 功能说明

序号	功能名称	功能描述
1	输出第二个运算数	C = B
2	或	C = A | B
3	减法	C = A - B
4	加法	C = A + B
5	比较	C = (A == B) ? 1 : 0(32 位)

（4）EXT

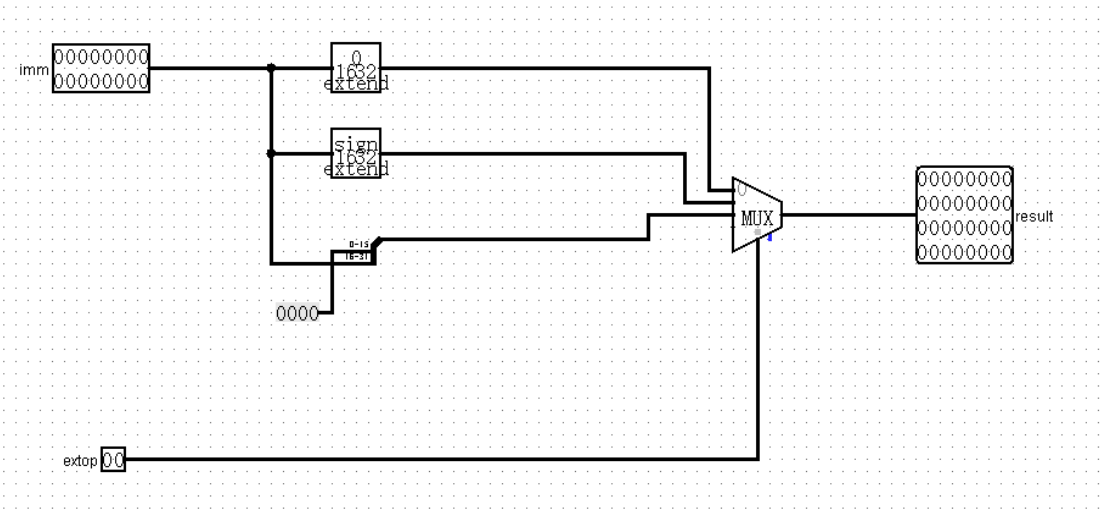


图 5-EXT 设计

表 7-EXT 模块说明

端口名称	方向	端口定义
imm[15:0]	I	16 位立即数
extop	I	扩展方式控制信号： 00：非符号扩展 01：符号扩展 10：低位补 0 扩展
result[31:0]	O	32 位扩展结果

表 8-EXT 功能说明

序号	功能名称	功能描述
1	非符号扩展	高 16 位补 0
2	符号扩展	高 16 位补符号
3	立即数加载至高位	立即数加载至高 16 位，低位补 0

（5）DM

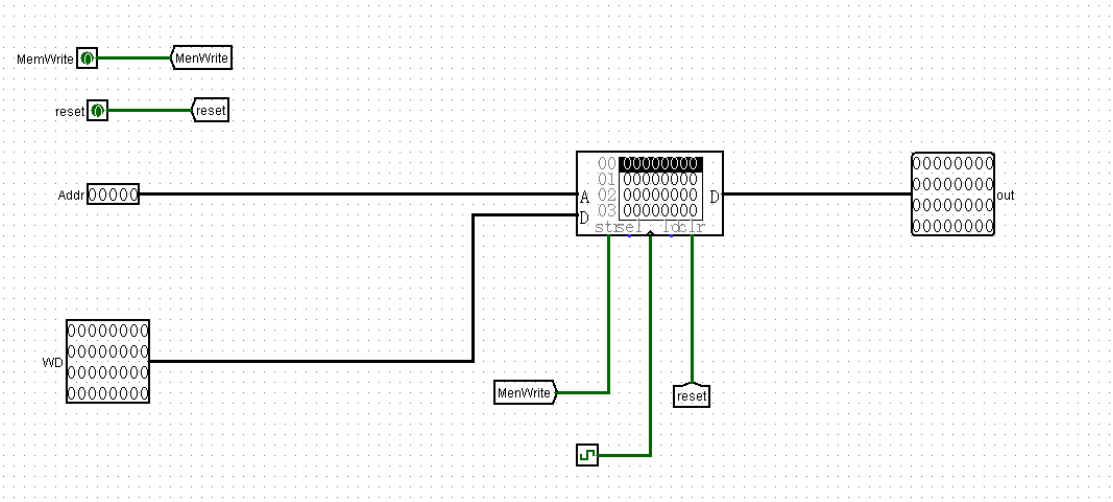


图 6-DM 设计

表 9-DM 模块说明

端口名称	方向	端口定义
MemWrite	I	写控制信号： 1：向 RAM 中写入数据 0：不向 RAM 中写入数据
reset	I	复位信号
Addr[4:0]	I	5 位操作寄存器地址
WD[31:0]	I	32 位写入数据
Out[31:0]	O	32 位输出数据

表 10-DM 功能说明

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	根据寄存器地址输出该寄存器中的数据
3	写	根据寄存器地址向该寄存器中写入数据

（6）control

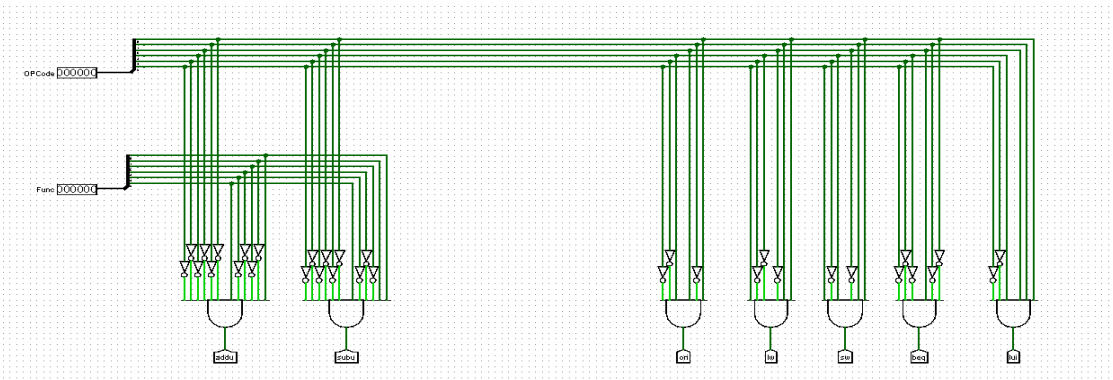


图 7-control 设计 1

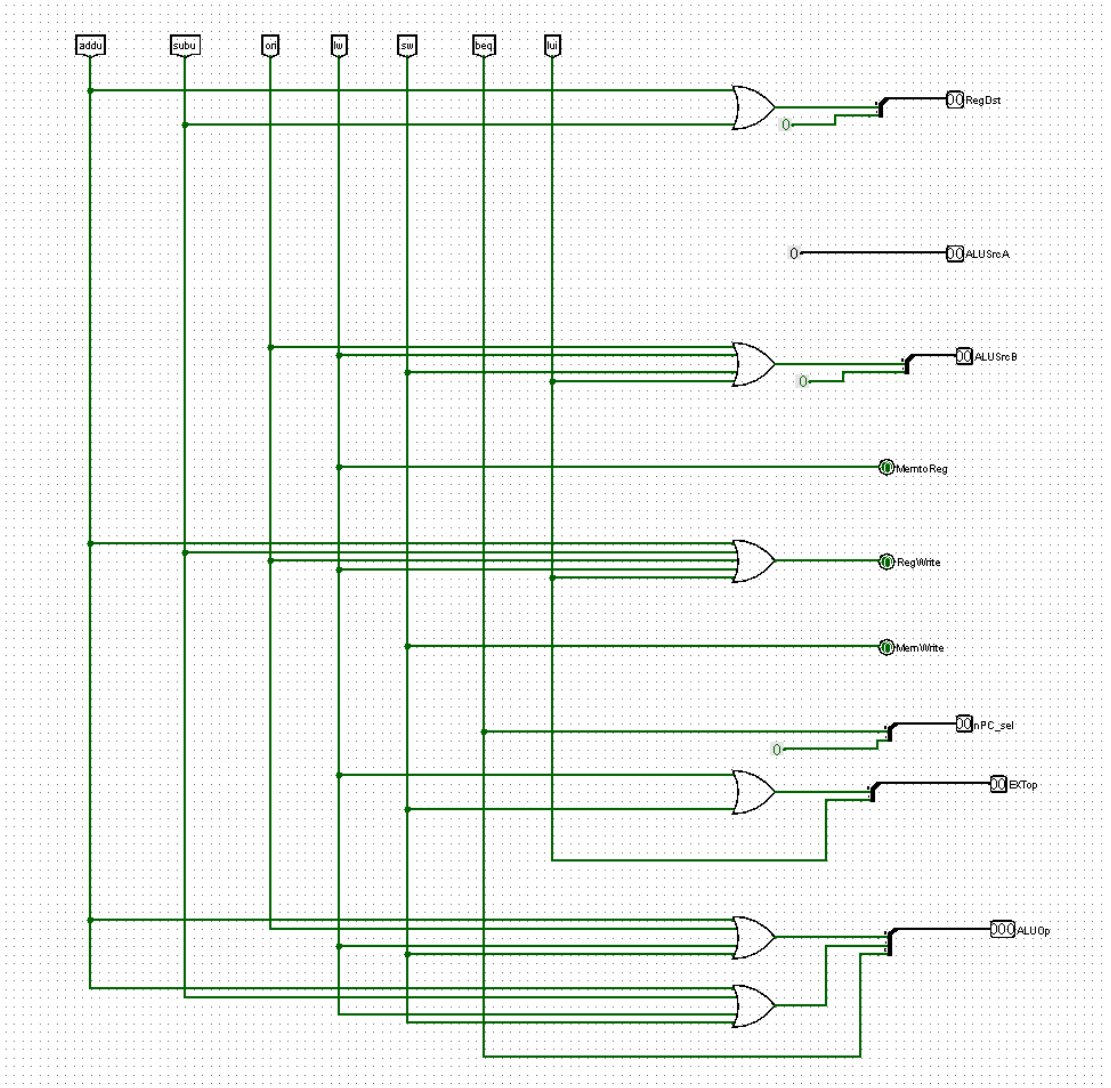


图 8-control 设计 2

表 11-control 模块说明

端口名称	方向	端口定义
OpCode[5:0]	I	写控制信号： 1：向 RAM 中写入数据 0：不向 RAM 中写入数据
Func[5:0]	I	复位信号
RegDst[1:0]	O	5 位操作寄存器地址
ALUSrcA[1:0]	O	选择ALU的第一个操作数 00：[rs]
ALUSrcB[1:0]	O	选择ALU的第二个操作数 00:[rt] 01:扩展的立即数
MemtoReg	O	选择写入寄存器堆的数据 0：来自 ALU 1：来自 DM
RegWrite	O	是否向寄存器堆中写入数据
MemWrite	O	是否向 DM 中写入数据
nPC_sel[1:0]	O	判断当前指令是否为 beq 00：不是 beq 01:是 beq
EXTop[1:0]	O	扩展方式控制信号： 00：非符号扩展 01：符号扩展 10：低位补 0 扩展
ALUOp[2:0]	O	控制信号： 000：将第二个运算数直接作为输出结果 001：或运算 010：减法运算 011：加法运算 100：判断是否相等

表 12-control 功能说明

序号	功能名称	功能描述
1	生成控制信号	根据当前指令生成相应的控制信号

三，控制器设计

表 13-指令识别

	addu	subu	ori	lw	sw	beq	lui
OpCode	000000	000000	001101	100011	101011	000100	001111
Func	100000	100011	n/a				

表 14-控制信号真值表

	RegDst	ALUSrcA	ALUSrcB	MemtoReg	RegWrite	MemWrite	nPC_sel
addu	01	00	00	0	1	0	00
subu	01	00	00	0	1	0	00
ori	00	00	01	0	1	0	00
lw	00	00	01	1	1	0	00
sw	x	00	01	x	0	1	00
beq	x	00	00	x	0	0	01
lui	00	00	01	0	1	0	00

	ExtOp	ALUOp
addu	x	011
subu	x	010
ori	00	001
lw	01	011
sw	01	011
beq	x	100
lui	10	000

四，测试程序设计

测试代码:

表 15-测试代码

源代码	对应机器码
lui $4,11111	3c042b67
ori $5,$0,9	34050009
beq $4,$0,jump1	10800002
addu $6,$5,$4	00a43021
addu $5,$6,$4	00c42821
jump1:	
subu $6,$4,$0	00803023
beq $5,$5,jump2	10a50001
ori $4,$5,0	34a40000
jump2:	
sw $4,0($3)	ac640000

sw $5,4($3)	ac650004
lw $5,0($3)	8c650000
nop	00000000
ori $4,$0,2018	340407e2
subu $6,$6,$4	00c43023
subu $5,$6,$4	00c42823
lui $7,12345	3c073039
ori $6,$4,54321	3486d431

Mars 期待结果：

Address	Value (+0)	Value (+4)	Value (+8)
0x00000000	0x2b670000	0x56ce0009	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000

图 9-MARS 期望 1

$a0	4	0x000007e2
$a1	5	0x2b66f03c
$a2	6	0x0000d7f3
$a3	7	0x30390000

图 10-MARS 期望 2

Logisim 仿真结果：

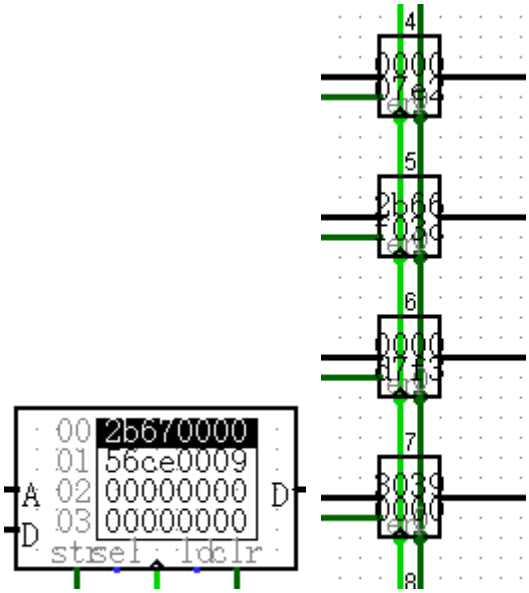


图 11-logisim 仿真结果

五， 思考题

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

答：优点：a）一定程序上节省空间

缺点：a）支持的指令个数减少 b）在 32 位系统中影响一些操作，例如 jal 将 PC+1 存入$31，而寄存器是 32 位的，需要补位不方便

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。

答：在 IM 中使用 ROM 将代码一次性导入，保证了程序的整体性，但是因为 ROM 的只读性假如我要临时修改代码就不太方便，可以考虑使用 RAM 方便对代码修改。在 DM 中使用 RAM 满足对特定地址的读写操作，简化了电路的设计难度。在 GRF 中使用寄存器堆,对不同地址的寄存器分开实现读写操作,暂无改进意见。

3,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。）

答：

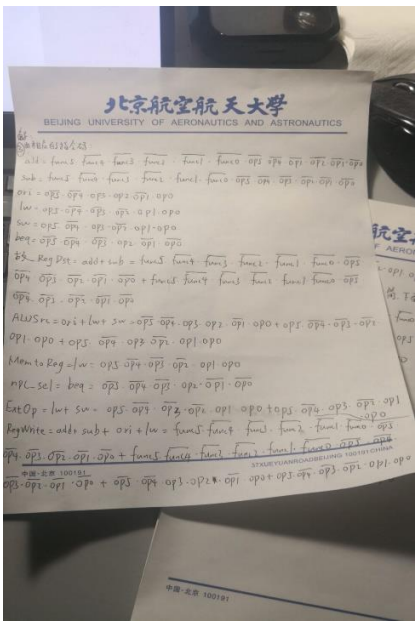


图 12-第三题答案

4,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

：

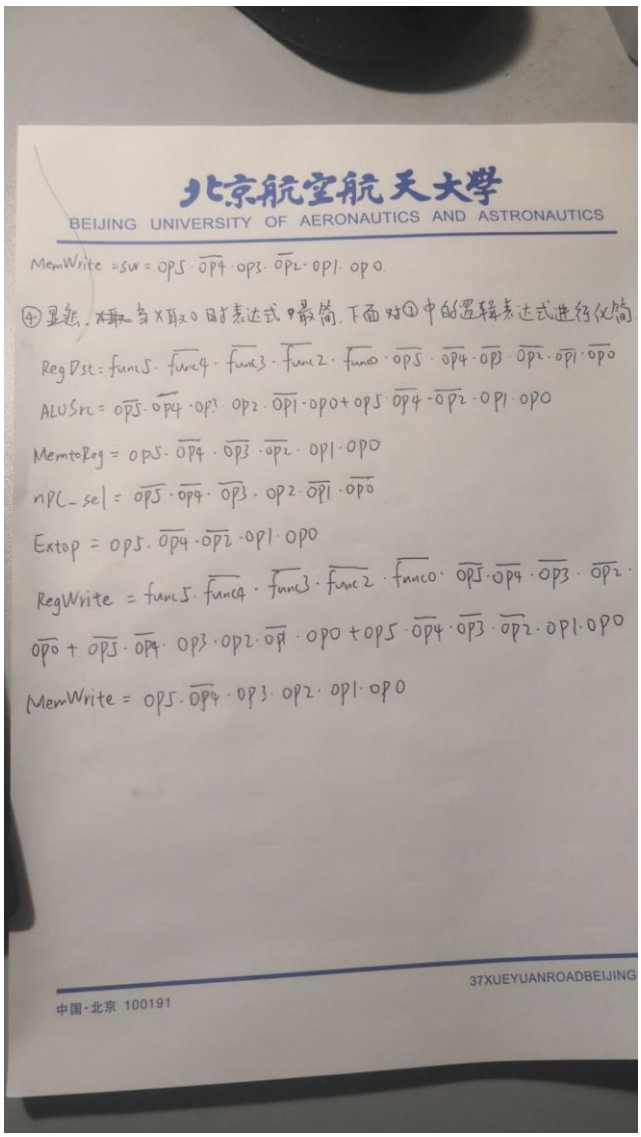


图 13-第四题答案

5，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：因为 nop 指令对各个部件都没有影响，它不对寄存器和内存读写以及 beq 指令产生任何影响，故不需要加入真值表。

6，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。

答：在 MARS 中是按字寻址，故占用 4 个字节，而在 logisim 中是按一位寻址的，所以只需对指令码中的数据偏移量除 4，即在这里用分离器取 2 到 6 位即可。

7，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

答：形式验证能够对于已有的设计和实现进行模型特性和数字逻辑的对比，根据某个或某些形式规范或属性，使用数学的方法直接证明其正确性或非正确性而不需要设计多样测试代码，就能实现对于实现和设计的一致性比对的优点。但是，形式验证同样具有无法察觉设计本身存在的固有缺陷。因此，形式验证补充了模拟验证的不足，二者各有优势，互为补充，缺一不可。在实际的检测中，我们应结合两种方法对电路进行充分的测试。