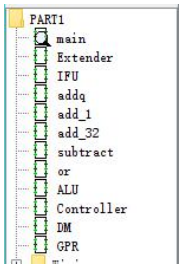
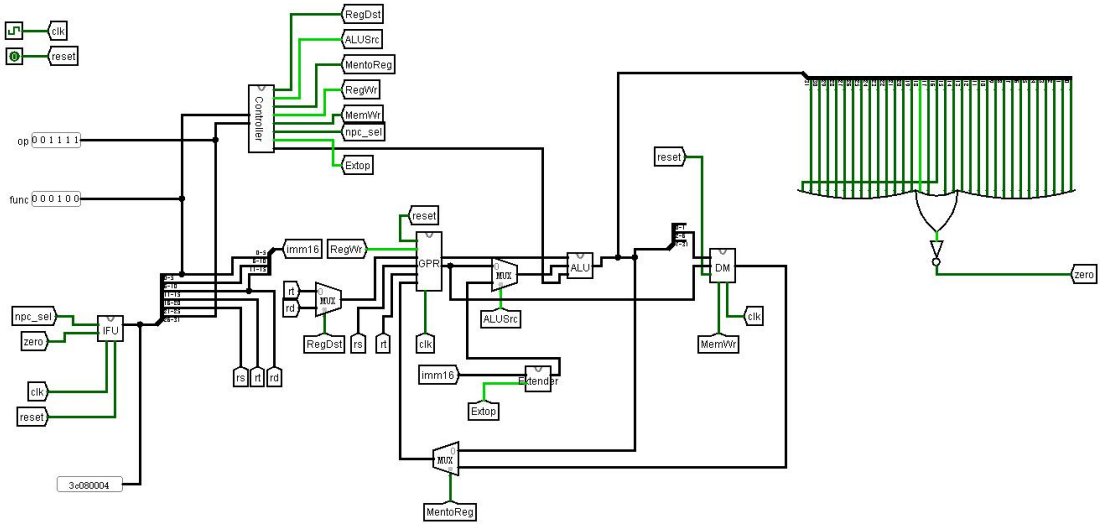
Project3 Logisim 完成单周期处理器开发实验报告

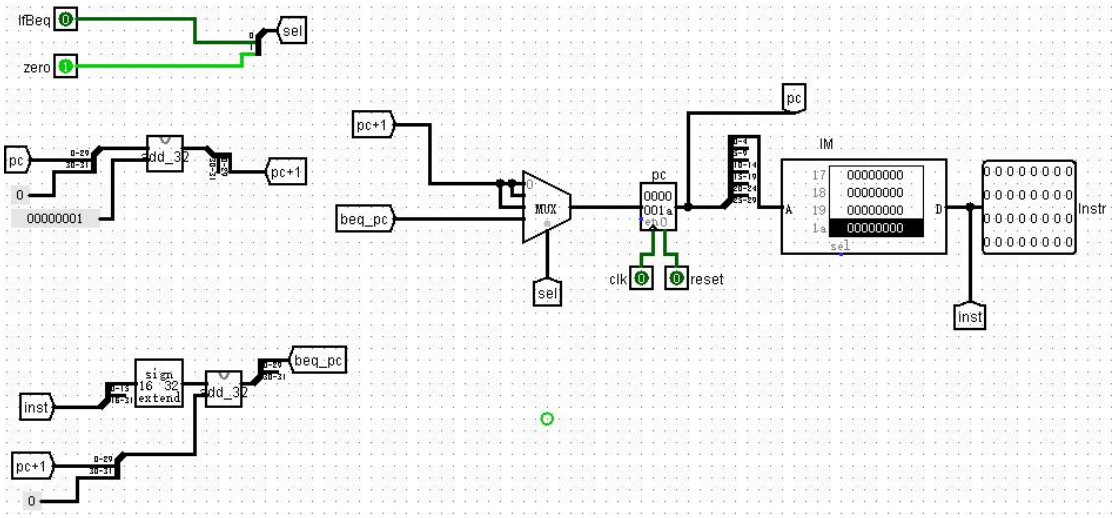
一．总体设计





二．模块定义

（1）IFU



信号名	方向	描述
IfBeq	I	当前指令是否为 beq 指令标志 1：当前指令为 beq 0：当前指令非 beq
Zero	I	ALU 计算结果为 0 标志 1：计算结果为 0 0：计算结果非 0
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
Instr[31:0]	O	32 位 MIPS 指令

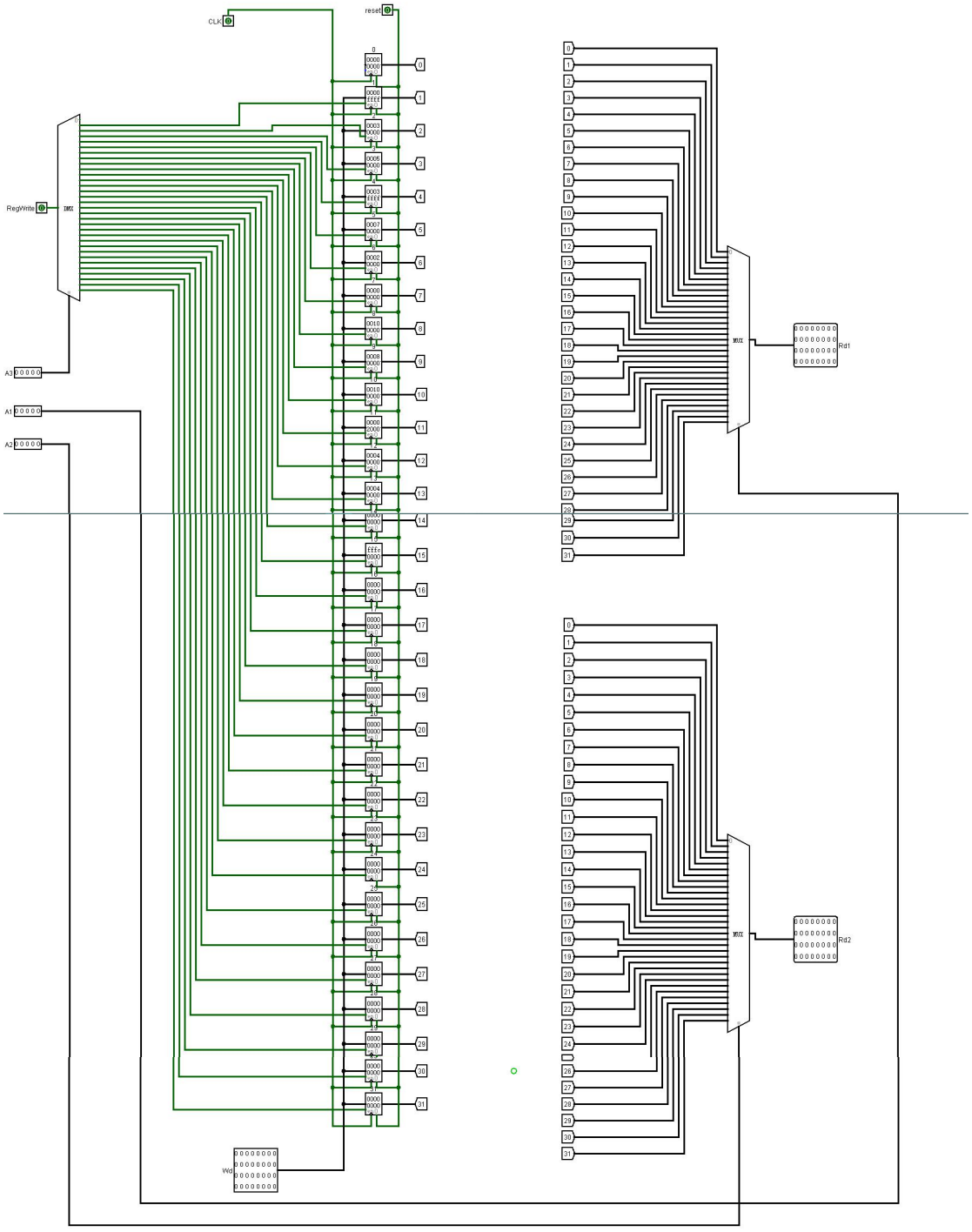
功能定义

序号	功能名称	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+1 如果当前指令是 beq 指令，并且 zero 为 1，则，PC<-PC+sign_ext

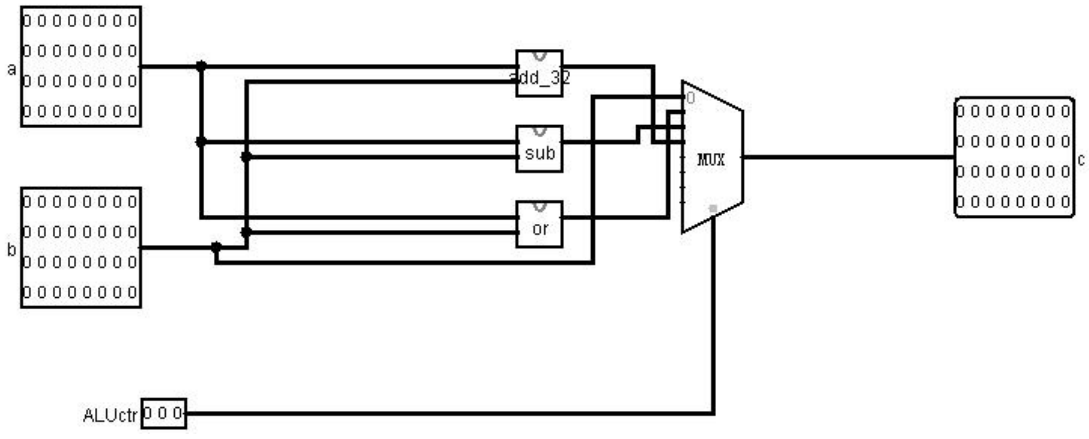
（2）GPR

模块接口

信号名	方向	描述
Wd[31:0]	I	写入数据的输入
Regwrite	I	读写控制信号 1：写操作 0：读操作
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
A1[4:0]	I	读寄存器地址 1
A2[4:0]	I	读寄存器地址 2
A3[4:0]	I	写寄存器地址
Rd1[31:0]	O	32 位数据输出 1
Rd2[31:0]	O	32 位数据输出 2


CLK
reset
RegWrite
A3
A1
A2
Rd1
Rd2
Wd

（3）ALU


a
b
add_32
sub
or
MUX
c
ALUctr

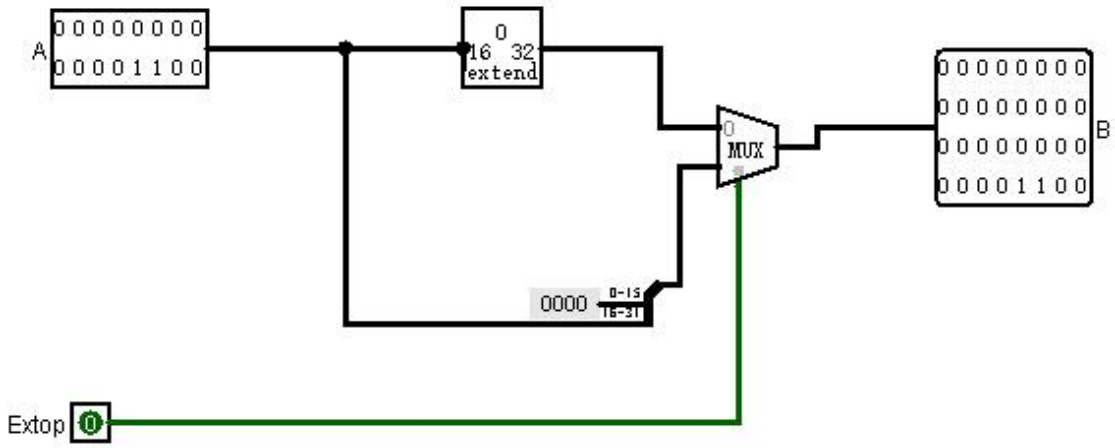
模块接口

信号名	方向	描述
A[31:0]	I	32 位输入数据 1
B[31:0]	I	32 位输入数据 2
F[1:0]	I	控制信号 01：或运算 10：减法
		11：加法
C[31:0]	0	32 位数据输出

功能定义

序号	功能名称	描述
1	或	A|B
2	减	A-B
3	加	A+B

（4）EXT



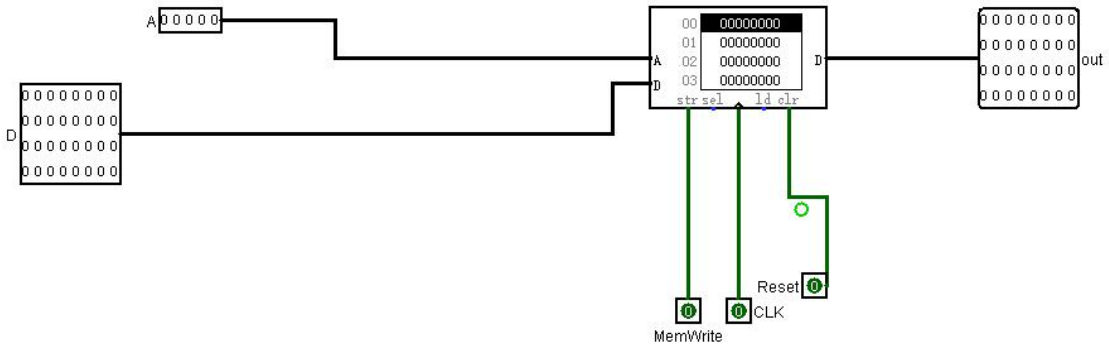
模块接口

信号名	方向	描述
A[15:0]	I	16 位数据输入
Extop	I	控制信号 0：高位补 0 1：低位补 0
B[31:0]	0	32 位数据输出

功能定义

序号	功能名称	描述
1	高位补 0	高 16 位补 0
2	低位补 0	低 16 位补 0

（5）DM



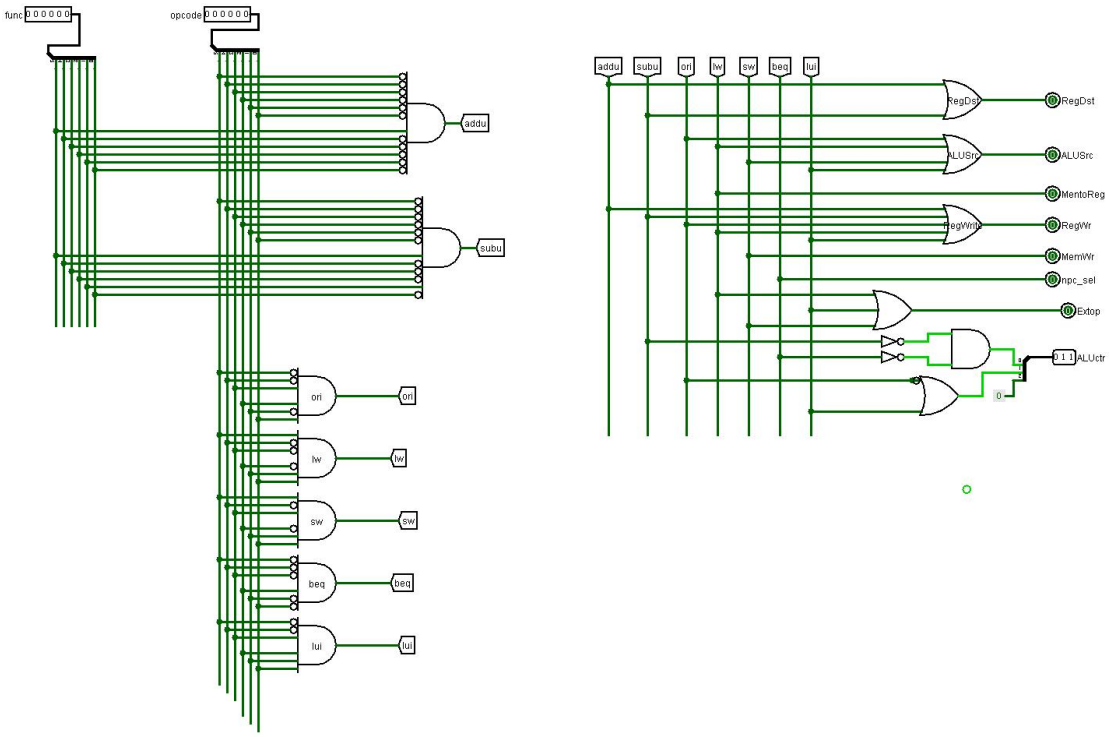
模块接口

信号名	方向	描述
D[31:0]	I	写入数据的输入
Memwrite	I	读写控制信号 1：写操作
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
A[4:0]	I	操作寄存器地址
out[31:0]	0	32 位数据输出

功能定义

序号	功能名称	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

（6）Controller



模块接口

信号名	方向	描述
op[5:0]	I	六位 op
func[5:0]	I	六位 function

Regdst	0	写地址控制
Alusrc	0	cpu 第二操作数选择控制
Memtoreg	0	DM 读控制
Regwrite	0	GPR 读写控制
Memwrite	0	DM 写控制，写入 GPR 数据选择
npc_sel	0	Beq 指令标志
extop	0	控制 ext 扩展方式
Aluop[1:0]	0	控制 cpu 进行相应运算

四．控制器设计

单周期真值表

	Func	100000	100010	N/A			
	Op	000000	000000	001101	100011	000100	001111
	add	sub	ori	lw	sw	beq	lui
RegDst	1	1	0	0	X	X	0
ALUSrc	0	0	1	1	1	0	1
MemtoReg	0	0	0	1	X	X	X
RegWrite	1	1	1	1	0	0	2
MemWrite	0	0	0	0	1	0	0
nPC_sel	0	0	0	0	0	1	0
ExtOp	X	X	0	0	0	X	1

ALUctr	Add	Subtract	Or	Add	Add	Subtract	X

五．测试要求

16.测试程序

```
lui $t0,0x0004                  #lui 测试程序要实现：立即数 0x0004 加载至 t0 寄存器的高位
lui $t1,0x0008                  #lui 测试程序要实现：立即数 0x0008 加载至 t1 寄存器的高位
ori $t3,$zero,0x00002000        #ori 测试程序要实现：zero 寄存器中的内容与立即数 0x00002000 进行或运算，储存在 t3 寄存器中
sw $t0,4($t3)                   #sw 测试程序要实现：把 t0 寄存器中值（1 Word），存储到 t3 的值再加上偏移量 4, 所指向的 RAM 中
sw $t0,8($t3)                   #sw 测试程序要实现：把 t0 寄存器中值（1 Word），存储到 t3 的值再加上偏移量 8, 所指向的 RAM 中
loop:add $t2,$t2,$t1                    #add 测试程序要实现：t1 寄存器中的值加上 t2 寄存器中的值后存到 t2 寄存器中
lw $t4,4($t3)                           #lw 测试程序要实现：把 t3 寄存器的值+4 当作地址读取存储器中的值存入 t4
lui $t5,0x0004                          #lui 测试程序要实现：立即数 0x0004 加载至 t5 寄存器的高位
sub $t7,$t6,$t5                         #sub 测试程序要实现：t6 寄存器中的值减去 t5 寄存器中的值后存到 t7 寄存器中
add $t0,$t0,$t5                         #sub 测试程序要实现：t0 寄存器中的值减去 t5 寄存器中的值后存到 t0 寄存器中
add $t6,$t6,$t0                         #add 测试程序要实现：t6 寄存器中的值加上 t0 后存到 t6 寄存器中
beq $t0,$t1,loop                        #beq 测试程序要实现：判断 t0 的值和 t1 的值是否相等，相等转 loop
add $t0,$t0,$t5                         #add 测试程序要实现：t0 寄存器中的值加上 t5 后存到 t0 寄存器中
lui $v0,0x0001                          #lui 测试程序要实现：立即数 0x0001 加载至 v0 寄存器的高位
lui $v1,0x0002                          #lui 测试程序要实现：立即数 0x0002 加载至 v1 寄存器的高位
add $v0,$v0,$v1                         #add 测试程序要实现：v0 寄存器中的值加上 v1 后存到 v0 寄存器中
add $v1,$v0,$v1                         #add 测试程序要实现：v0 寄存器中的值加上 v1 后存到 v1 寄存器中
ori $a0,$v0,0xffff                      #ori 测试程序要实现：v0 寄存器中的内容与立即数 0xffff 进行或运算，储存在 a0 寄存器中
sub $a1,$a0,0x0000ffff                  #sub 测试程序要实现：a0 寄存器中的值减去立即数 0x0000ffff 后存到 a1 寄存器中
loop2:sub $a2,$v1,$v0                   #sub 测试程序要实现：v1 寄存器中的值减去 v0 中的值后存到 a2 寄存器中
add $a1,$a2,$a1                         #add 测试程序要实现：a2 寄存器中的值加上 a1 后存到 a1 寄存器中
beq $a1,$v1,loop2                       #beq 测试程序要实现：判断 a1 的值和 v1 的值是否相等，相等转 loop2
```

机器码：

```
3c080004 3c090008 340b2000 ad680004 01495020 8d6c0004 3c0d0004 01cd7822 010d4020 01c87020 1109fff9
010d4020 3c020001 3c030002 00431020 00431820 3444ffff 3c010000 3421ffff 00812822 00623022 00c52820
10a3fffd
```

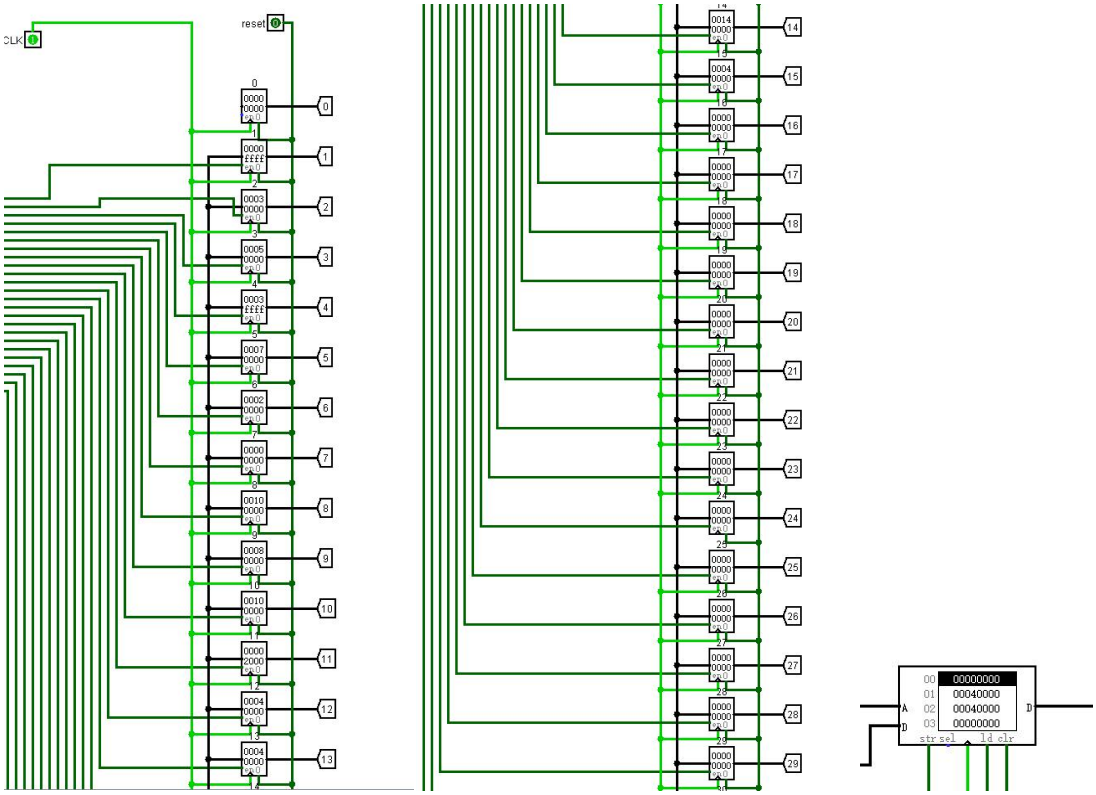
MARS 模拟结果：

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x0000ffff
$v0	2	0x00030000
$v1	3	0x00050000
$a0	4	0x0003ffff
$a1	5	0x00070000
$a2	6	0x00020000
$a3	7	0x00000000
$t0	8	0x00100000
$t1	9	0x00080000
$t2	10	0x00100000
$t3	11	0x00002000
$t4	12	0x00040000
$t5	13	0x00040000
$t6	14	0x00140000
$t7	15	0x00040000
$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	0x00003ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000005c
hi		0x00000000
lo		0x00000000

Logism：

GPR: DM:



六、 问答

18. 对于 Figure5、 Figure6 中的与或阵列来说， 1 个 3 输入与门最终转化为 2 个 2 输入与门， 1 个 4 输入与门最终转化为 3 个 2 输入与门，依次类推。或阵列也类似计算。那么

a) 请给出采用 Figure5、 Figure6 中的方法设计的每个控制信号所对应的 2 输入与门、 2 输入或门、非门的数量。

19.	2 输入与门	2 输入或门	非门

RegDst	17	1	10
RegWrite	32	4	10
ALUsrc	20	3	5
PCsrc	5	0	5
MemWrite	5	0	2
MemRead	5	0	3
MemtoReg	5	0	3
ExtOp	15	2	4
ALUctr[1]	10	1	3
ALUctr[0]	21	2	9

a) 请与第 17 项对比，你更喜欢哪种设计方法。为什么

第一种的每个控制信号都需要对其分配单独的与门、或门，因为它是直接对 op、func 的 12 位或 6 位信号的逻辑表达式，所以没有针对性并且浪费元件。

而第二种是先把 op、func 变成相应的指令信号，再由指令信号生成控制信号。当一种指令对应多种控制信号为 1 时，不必再对每个信号再单独为这个指令分配与门，而可以共用这个指令的信号，再添加或门就可以了。