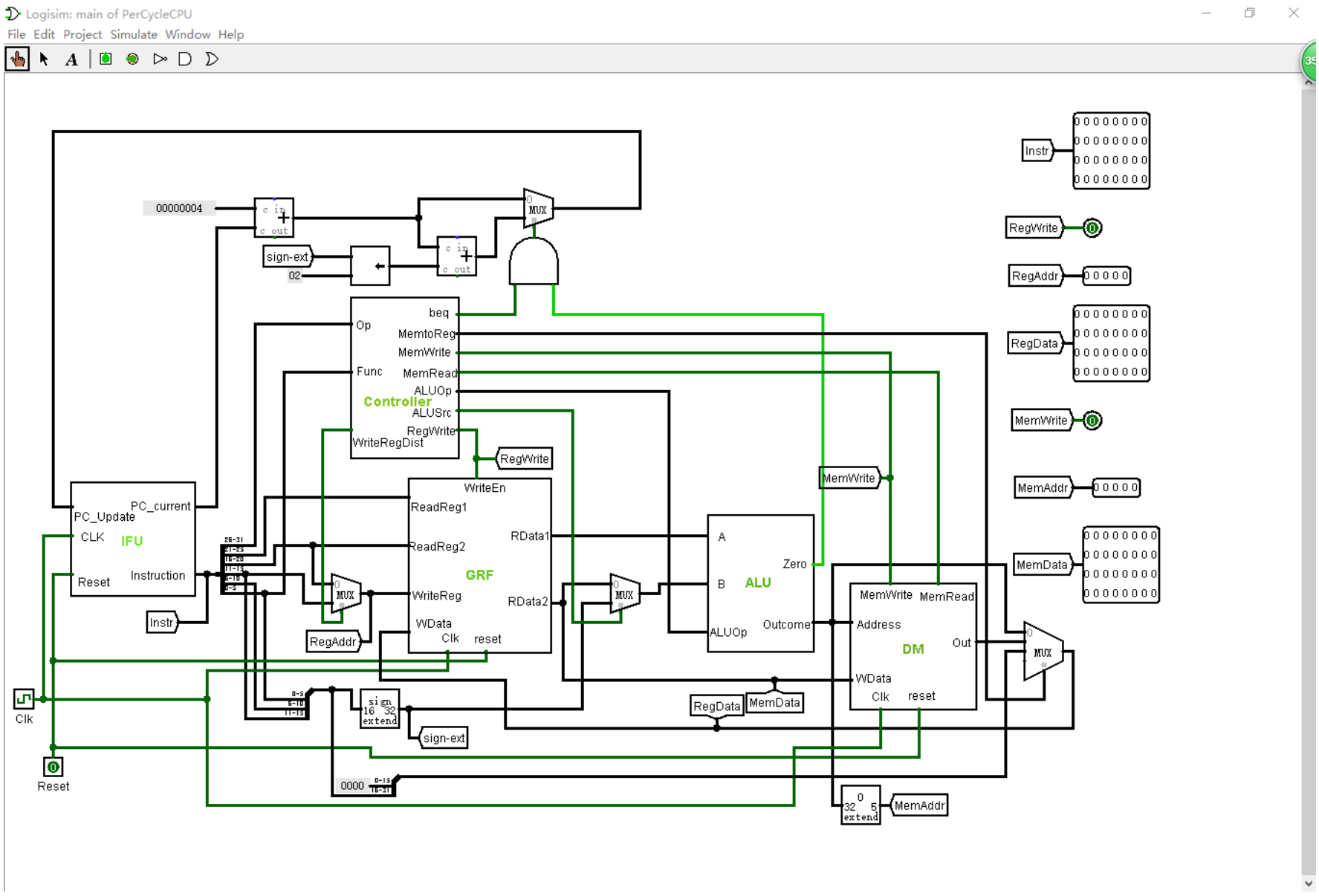
利用 logisim 开发支持八条指令的 MIPS 单周期处理器

一、 设计概述:

本实验利用 logisim 开发支持 MIPS 指令集中的 addu、subu、ori、lw、sw、lui、beq、nop 八条指令的单周期处理器。本实验设计的单周期 CPU 包含 Controller(控制器)、IFU(取指令单元)、GRF(通用寄存器组/寄存器文件/寄存器堆)、ALU(算术逻辑运算单元)、DM(数据存储器)、EXT(位扩展)等基本部件，借助多路选择器 Multiplexer 和译码器 Decoder 以及 Splitter 等 logisim 内置器件组合连接成 DataPath。



Logisim 搭建的 CPU 顶层设计图

数据通路设计表格

指令	Adder		PC	IM.Address	Registers				ALU		DM		Sign-ext	NAdd	
	A	B			Reg1	Reg2	Wreg	WData	A	B	Address	WData		A	B
addu	PC	4	Adder	PC	Rs	Rt	Rd	ALU	RData1	RData2					
subu	PC	4	Adder	PC	Rs	Rt	Rd	ALU	RData1	RData2					
ori	PC	4	Adder	PC	Rs		Rt	ALU	RData1	Sign-ext			Imm16		
lw	PC	4	Adder	PC	Rs		Rt	DM	RData1	Sign-ext	ALU		Imm16		
sw	PC	4	Adder	PC	Rs	Rt			RData1	Sign-ext	ALU	RData2	Imm16		
lui	PC	4	Adder	PC			Rt	Sign-ext					Imm16		
beq	PC	4	NAdd	PC	Rs	Rt			RData1	RData2			Imm16	Adder	Shift
nop	PC	4	Adder	PC											
合并	PC	4	Adder/NAdd	PC	RS	Rt	Rt/Rd	ALU/DM/Sign-ext	RData1	RData2/Sign-ext	ALU	RData2	Imm16	Adder	Shift

二、 模块定义：

本实验设计的 CPU 共有 Controller、GRF、IFU、ALU、EXT、DM 六个模块。下面给出各模块的模块接口以及功能定义。

a) IFU

模块接口：

信号名	方向	描述
Reset	I	复位信号 1:复位 0：无效
Clk	I	时钟信号
PC_Update[31:0]	I	PC 寄存器的输入端口，时钟信号有效时更新 PC 的值
PC_current[31:0]	O	当前 PC 的值，指令寄存器 IM 的地址输入端
Instruction[31:0]	O	32 位的 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<-0x00000000
2	取指令	依据 PC[6:2],从 IM 取出指令
3	更新 PC	时钟上升沿(有效),PC 寄存器写入新的 PC 值,获得下一条指令地址

b) GRF

模块接口:

信号名	方向	描述
ReadReg1[4:0]	I	读寄存器地址 1
ReadReg2[4:0]	I	读寄存器地址 2
WriteReg[4:0]	I	写寄存器地址
WriteEn	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
WData[31:0]	I	写入数据的输入
RData1[31:0]	O	32 位输出数据 1
RData2[31:0]	O	32 位输出数据 2

功能定义：

序号	功能名称	描述
1	复位	当复位信号有效时，GRF 的所有寄存器值 0x00000000
2	读寄存器	根据读寄存器的输入地址读出数据
3	写寄存器	根据写寄存器的输入地址，在写信号有效时，把输入的数据写入指定寄存器中

c) ALU

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
ALUOp[1:0]	I	ALU 控制信号 00：32 位加法 01:32 位减法 10：32 位或运算
Zero	O	两个 32 位数据是否相等
Outcome[31:0]	O	32 位数据输出

功能定义：

序号	功能名称	描述
1	加	A+B
2	减	A-B
3	或	A|B
4	相等	A==B

d) EXT(使用 logisim 内置的 BitExtender)

模块接口：

信号名称	方向	描述
Imm[15:0]	I	16 位数据输入
Ext_Outcome[31:0]	O	32 位数据输出

功能定义：

序号	功能名称	描述
1	符号扩展	最高位为 1，则新增 16 位作为高 16 位全部置 1 最高位为 0，新增 16 位作为高 16 位全部置 0
2	加载至高 16 位	新增 16 位作为次 16 位，全部置 0

e) DM

模块接口：

信号名称	方向	描述
Addr[31:0]	I	操作寄存器地址(取其[4:0]部分)
WData[31:0]	I	写入存储器的数据输入
MemWrite	I	写控制信号 1：写操作
MemRead	I	读控制信号 1：读操作
Clk	I	时钟信号
Reset	I	复位信号 1：复位 0：无效
Out	O	读有效时输出的数据

功能定义：

序号	功能名称	描述
1	复位	当复位信号有效时，数据存储器所有值置为 0x00000000
2	读取	操作信号有效时，读取内存数据
3	写入	写操作信号有效时，数据写入内存

f)Controller

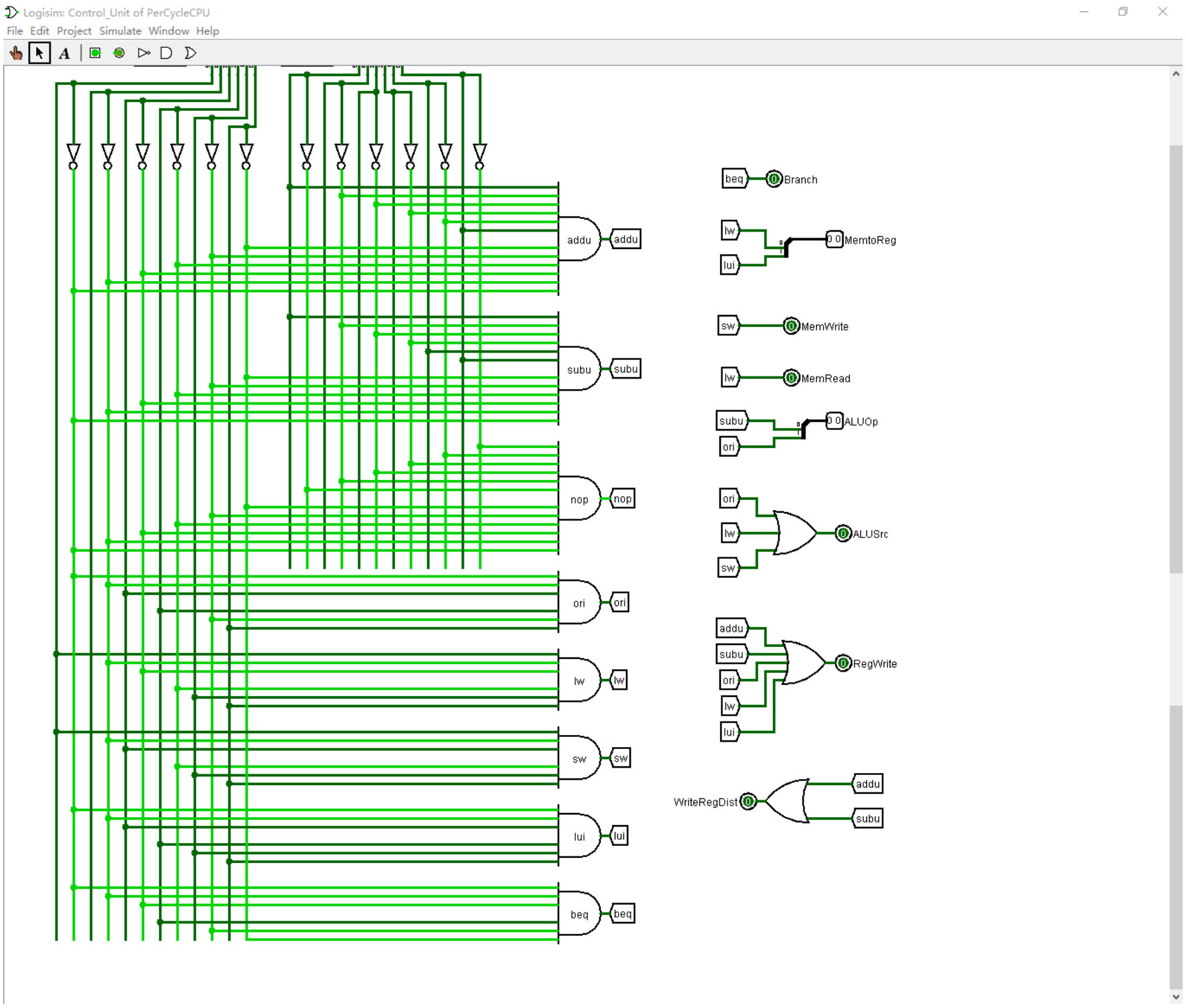
模块接口：

信号名称	方向	描述
Op[5:0]	I	六位 Operation code
Func[5:0]	I	六位 Function code
Branch	O	Beq 指令分支信号
MemtoReg[1:0]	O	写入 GRF 的数据选择信号
MemWrite	O	写 DM 的控制信号
MemRead	O	读 DM 的控制信号
ALUOp[1:0]	O	ALU 的运算操作模式选择信号
ALUSrc	O	ALU 的 B 端数据来源选择信号
RegWrite	O	GRF 写入数据控制信号

三、 控制器设计

主控单元真值表

Func	100001	100011						000000
OP	000000	000000	001101	100011	101011	001111	000100	000000
指令	Addu	Subu	Ori	Lw	Sw	Lui	Beq	nop
Branch	0	0	0	0	0	0	1	0
MemtoReg[1:0]	00	00	00	01	xx	10	xx	xx
MemWrite	0	0	0	0	1	0	0	0
MemRead	0	0	0	1	0	0	0	0
ALUOp[1:0]	add(00)	sub(01)	or(10)	add(00)	add(00)	xx	xx	xx
ALUSrc	0	0	1	1	1	x	0	x
RegWrite	1	1	1	1	0	1	0	0
WriteRegDist	1	1	0	0	x	0	x	x



Logisim 搭建的 Controller

逻辑真值表达式：

1. 与逻辑

$$\text{addu} = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ \overline{Func1}\ Func0$$

$$\text{subu} = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ Func1\ Func0$$

$$\text{ori} = \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ \overline{Op1}\ Op0$$

$$\text{lw} = Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$$

$$\text{sw} = Op5\ \overline{Op4}\ Op3\ \overline{Op2}\ Op1\ Op0$$

$$\text{lui} = \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ Op1\ Op0$$

$$\text{beq} = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ Op2\ \overline{Op1}\ \overline{Op0}$$

$$\text{nop} = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ \overline{Func5}\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ \overline{Func1}\ \overline{Func0}$$

2. 或逻辑

$$\text{Branch} = \text{beq}$$

$$\text{MemtoReg} = \{lui, lw\}\ \text{//\{,\}是位拼接}$$

$$\text{MemWrite} = \text{sw}$$

$$\text{MemRead} = \text{lw}$$

$$\text{ALUOp} = \{\text{ori}, \text{subu}\}$$

$$\text{ALUSrc} = \text{ori} + \text{lw} + \text{sw}$$

$$\text{RegWrite} = \text{addu} + \text{subu} + \text{ori} + \text{lw} + \text{lui}$$

WriteRegDist = addu + subu

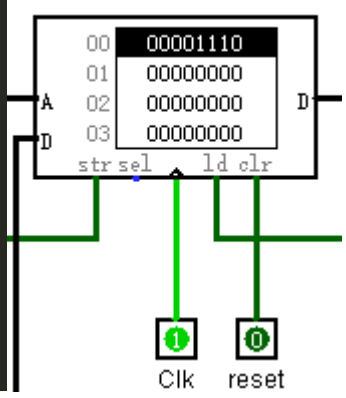
四、 测试 CPU

```
.data
    stack:.space 4 #声明4个字节的空间
.text
lui $t0,0x1000 #加载0x1000至最高16位，即$t0 <-- 0x10000000
ori $t0,0x0001 #或上0x0001，$t0 <-- 0x10000001
nop #空指令
addu $t1,$t0,$zero #无符号加，$t1 <-- $t0 + $zer0 期望$t1 <--0x10000001
ori $t1,0x1110 #或上立即数 $t1 <-- 0x10001111
subu $t2,$t1,$t0 #无符号减，$t2 <-- $t1 - $t0  期望$t2 <-- 0x00001110
nop #空指令
lui $t3,0x0000 #make $t3 为 0x00000000
label1:  #label1 作为标签，作为后面beq的跳转使用
    sw $t2,stack($zero) #数据写入内存 mem[$zero + stack] <-- $t2 ,期望写入0x00001110
    addu $t3,$t3,$t2  #根据下一条beq指令，程序执行两次此指令,第一次，期望$t3为0x00001110,第二次$t3为0x00002220
    beq $t2,$t3,label1 #if $t2==$t3,程序跳转到label1处(期望程序跳转一次label1，之后不再跳转)
    nop #空指令
lw $t2,stack($zero) #update $t2, $t2 <--mem[$zero +stack],期望$t2 <-- 0x00002220
nop #空指令
label2:beq $t2,$t2,label2 #if $t2==$t2,程序跳转label2,期望程序一直执行此条指令
```

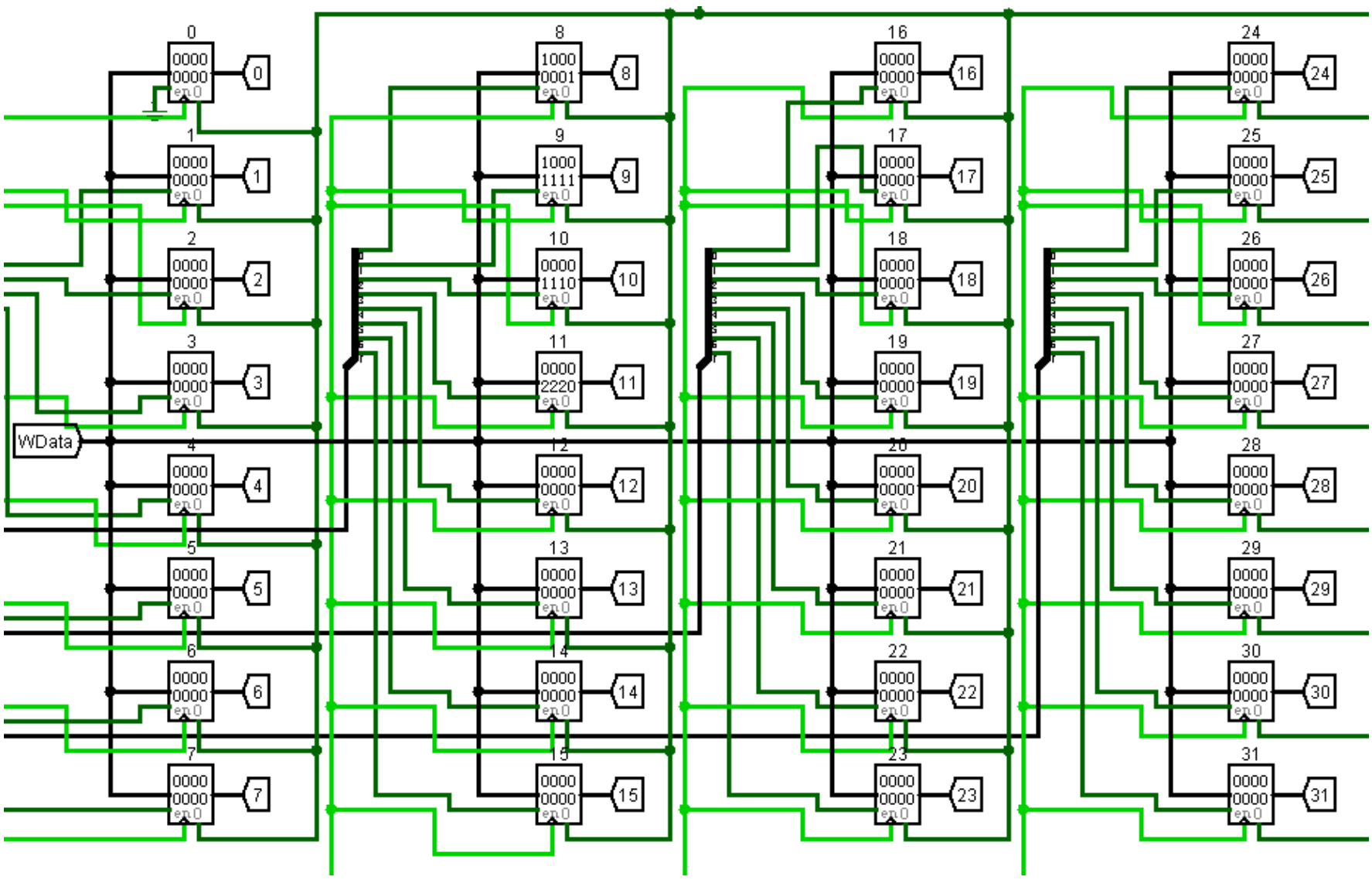
测试汇编程序(MIPS 指令)

```
00000000
3c081000
35080001
00000000
01004821
35291110
01285023
00000000
3c0b0000
ac0a0000
016a5821
114bfffd
00000000
8c0a0000
00000000
114affff
```

测试机器码



运行后 DM 的数据存储情况



运行后各寄存器的值

测试程序功能：

1. nop //空指令，PC+4
2. lui $t0,0x1000 //加载立即数至最高 16 位,往$t0 寄存器里写入 0x10000000
3. ori $t0,0x0001 //或立即数,ALU 进行按位或逻辑运算,往$t0 写入数据 0x10000001
4. nop //空指令
5. addu $t1,$t0,$zero //无符号加法,ALU 运算$t0 里的数值加上$zero 固定的数值 0,运算结果写入$t1
6. ori $t1,0x1110 //ALU 进行或运算,往$t1 写入数据 0x10001111
7. subu $t2,$t1,$t0 //ALU 进行减法运算,将$t1 寄存器的数值减去$t0 寄存器的数值,得到的结果写入$t2 寄存器,应该向$t2 寄存器写入数据 0x00001110
8. nop //空指令
9. lui $t3,0x0000 //初始化$t3,向$t3 寄存器写入数据 0
10. label1: sw $t2,stack($zero) //向内存地址为 32ext($zero+stack)的内存(DM)中写入$t2 寄存器的数值 0x00001110
11. addu $t3,$t3,$t2 //ALU 进行加法运算,GRF[t3]+GRF[t2]→GRF[t3]
12. beq $t2,$t3,label1 //ALU 判断 GRF[t2]==GRF[t3]? If is true,goto label1;else PC+4
13. lw $t2,stack($zero) //load 指令,加载 DM[stack+$zero]到 GRF[t2]
14. nop //空指令
15. label2:beq $t2,$t2,label2 //ALU 判断相等结果为 1,程序将会一直执行此条指令,PC 寄存器值不会发生变化,Controller 输出的所有控制信号将保持不变

最终程序运行后所有寄存器的值以及 DM 中的值应与图所示相对.

五、 思考题

1. 在上个学年的计组课程中，PC（程序计数器）位数被规定为30 位，试分析其与32 位PC 的优劣。
   答：32位PC可以访问4G的指令空间(PC+4),比30位的访问空间多。但是32位的PC设计上会比30位PC的设计更加复杂。逻辑电路会复杂化。由于32位PC的低两位都是0，所以如果用30位PC的话，每次PC+1即可，即将30位看作是32位PC的高30位
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：比较合理。PC取指令，指令对应完整的可执行的程序，IM里的值是不能被改变的，所以IM应该用ROM；DM存储操作数据，可能会有更新，所以用RAM。GRF是临时存储单元，所以用寄存器堆。暂无改进意见。若是多周期，可以将IM与DM合并在一起,用一个存储器，加上选择信号即可。
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。）
   $RegDst = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ Func0$
   $RegWrite = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ Func0 + \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ Op1\ Op0 + Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$
   $ALUSrc = \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ Op0 + Op5\ \overline{Op4}\ \overline{Op2}\ Op1\ Op0$
   $MemtoReg = Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$
   $nPC\_Sel = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ Op2\ \overline{Op1}\ \overline{Op0}$
   $ExtOp = Op5\ \overline{Op4}\ \overline{Op2}\ Op1\ Op0$
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。

   (第三题已经化为最简)

   $RegDst = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ Func0$
   $RegWrite = \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}\ Func5\ \overline{Func4}\ \overline{Func3}\ \overline{Func2}\ Func0 + \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ Op1\ Op0$

$$+\ Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$$

$ALUSrc = \ \overline{Op5}\ \overline{Op4}\ Op3\ Op2\ Op0 + Op5\ \overline{Op4}\ \overline{Op2}\ Op1\ Op0$

$MemtoReg = Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$

$nPC\_Sel = \ \overline{Op5}\ \overline{Op4}\ \overline{Op3}\ Op2\ \overline{Op1}\ \overline{Op0}$

$ExtOp = \ Op5\ \overline{Op4}\ \overline{Op2}\ Op1\ Op0$

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：nop指令机器码全是0。在nop执行的时候，类似MemWrite,WriteEn等要么是0，要么是x或者xx，化简的时候不需要考虑nop指令这一列。

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：如果假设DM有256MB，并且映射在0x30000000~0x3FFFFFFF区间,那么我们在做选择的时候，只需要当前指令取出来经过ALU运算后得到的地址的高4位与0x3作比较(无符号)即可,大于则片选信号为1，小于则为0。

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：形式验证优势：1.由于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，因此测试者不必考虑如何获得测试向量。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，因此有效地克服了模拟验证的不足。3. 形式验证可以进行从系统级到门级的验证，而且验证时间短，有利于尽早、尽快地发现和改正电路设计中的错误，有可能缩短设计周期。

   形式验证的劣势：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。

所以目前需要形式验证与模拟测试验证的综合。