Porject 3 logisim设计单周期cpu实验报告

一、整体介绍

1.设计的为32位单周期cpu

2.支持7条指令，分别为**{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。

**5.** 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、模块介绍

1.GRF

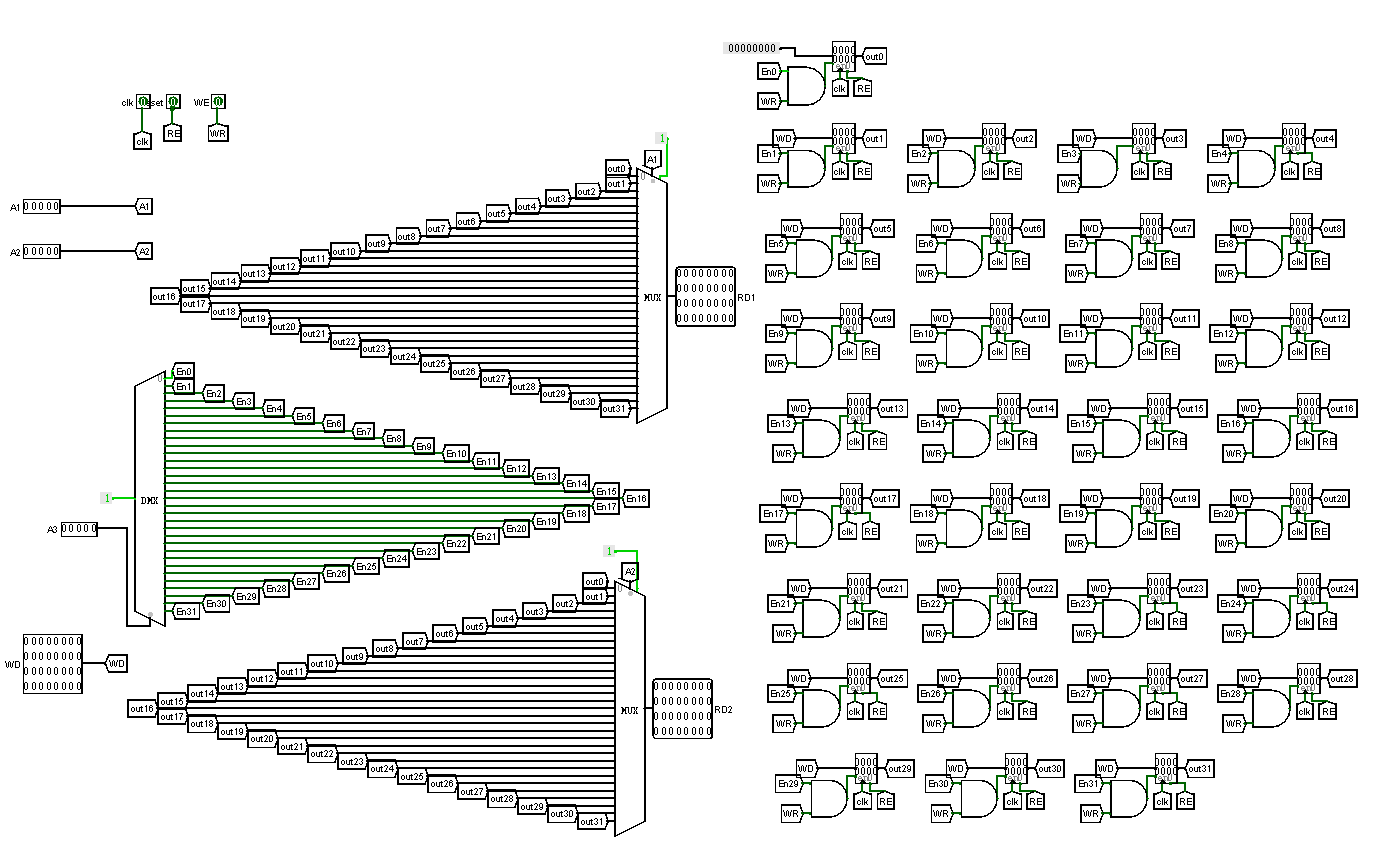
寄存器堆IO说明

|  |  |  |
| --- | --- | --- |
| 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 |
| RegWrite | I | 是否可以写入寄存器堆的控制信号  其中，1：可以写入  0：不可写入 |
| RA1[4:0] | I | 读操作寄存器地址1 |
| RA2[4:0] | I | 读操作寄存器地址2 |
| WA[4:0] | I | 写操作寄存器地址 |
| RegData [31:0] | I | 写入“写入寄存器”的内容 |
| RD1[31:0] | O | 读入寄存器1中的内容 |
| RD2[31:0] | O | 读入寄存器2中的内容 |

GRF功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset=1时，寄存器被置为 0x00000000 |
| 2 | 读寄存器 | 根据读入寄存器地址，读出寄存器内容 |
| 3 | 写寄存器 | 根据写入寄存器地址和写入信号，向寄存器内写入内容 |

实现图片



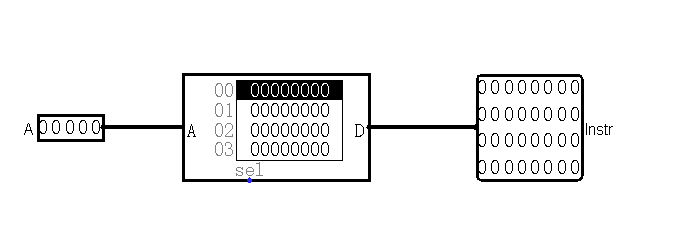
2.IM

IM接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| A[4:0] | I | 读入的5位地址 |
| Instr[31:0] | O | 输出的32位指令 |

IM功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指令 | 通过pc传入的5位地址，取出32位指令 |



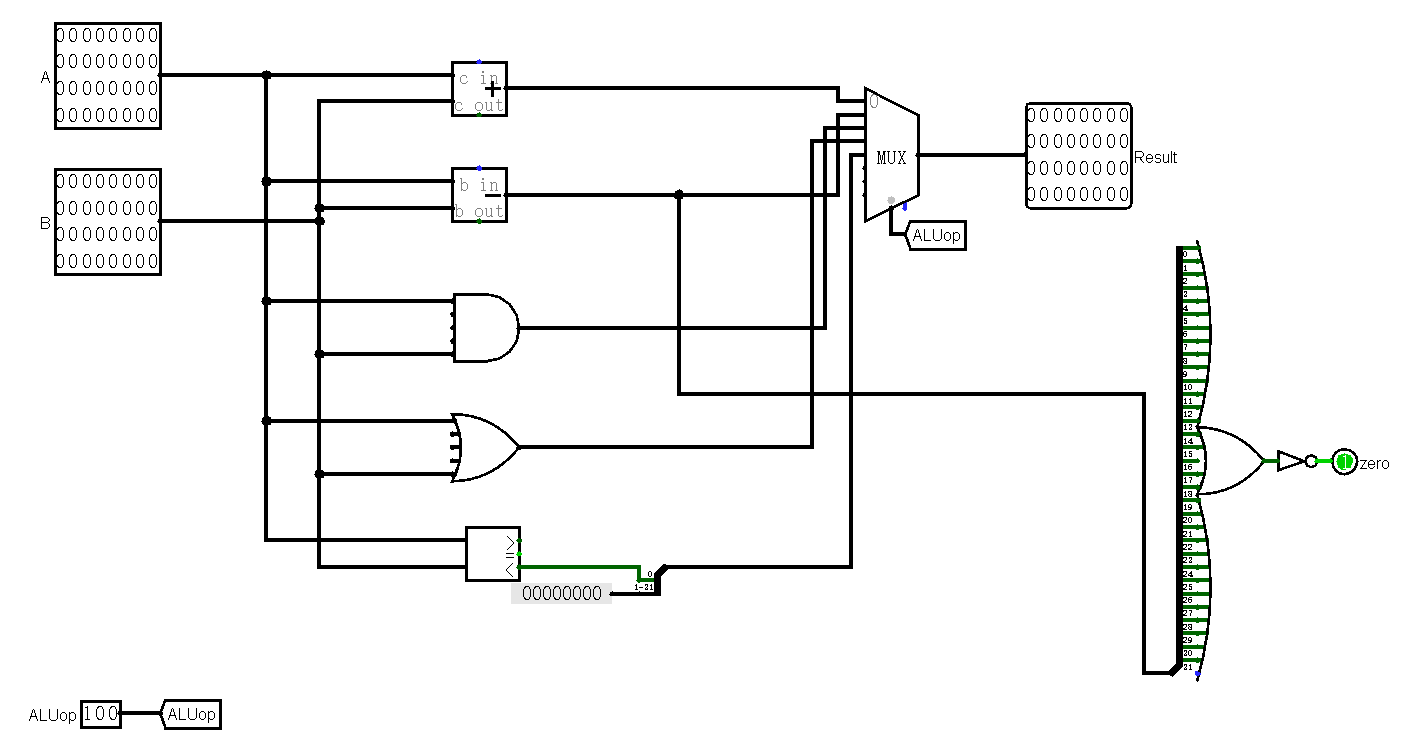
3.ALU

ALU接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| A[31:0] | I | 输入数据1 |
| B[31:0] | I | 输入数据2 |
| ALUop[2:0] | I | 输入选择信号  000:做加运算  001:做减运算  010:做与运算  011:做或运算  100:做比较运算 |
| Zero | O | 当 zero = 0 时，表示A = B  当 zero = 1 时，表示A != B |
| Result[31:0] | O | 输出 A与B做运算的32位结果 |

ALU功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | Result = A + B |
| 2 | 减法 | Result = A – B |
| 3 | 与 | Result = A & B |
| 4 | 或 | Result = A | B |
| 5 | 判断大小 | 若A = B,则 zero = 1, Result = 1  若 A < B, 则 zero = 0, Result = 1  若A > B, 则zero = 0, Result = 1 |



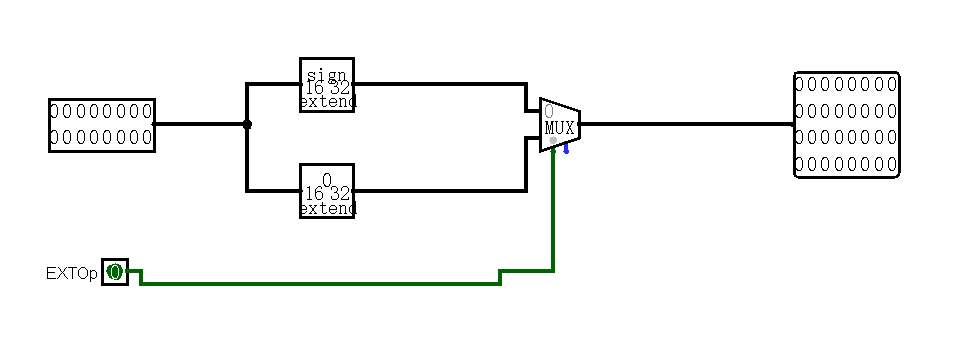
4.EXT

EXT接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| in[15:0] | I | 读入的16位待扩展数 |
| ExtOp | I | 扩展方式选择  0:有符号扩展  1:无符号扩展 |
| Out[31:0] | O | 输出的32位扩展结果 |

EXT功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 进行有符号扩展 | 将In[15]从Out[15]扩展到Out[31] |
| 2 | 进行无符号扩展 | 将Out[16]到Out[31]补0 |



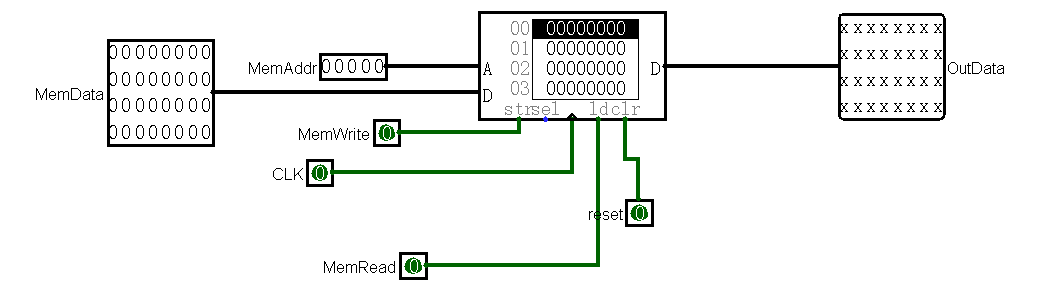
5.DM

DM接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位标志 |
| MemAddr[4:0] | I | 输入数据存储器的地址 |
| MemWrite | I | 写入控制信号 |
| MemRead | I | 执行读操作 |
| MemData[31:0] | I | 写入数据存储器的内容 |
| OutData[31:0] | O | 输出的32位数据 |

DM功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 将DM数据设置为0X00000000 |
| 2 | 读 | 读入DM内数据 |
| 3 | 写 | 向DM内写入32位数据 |



6.IFU

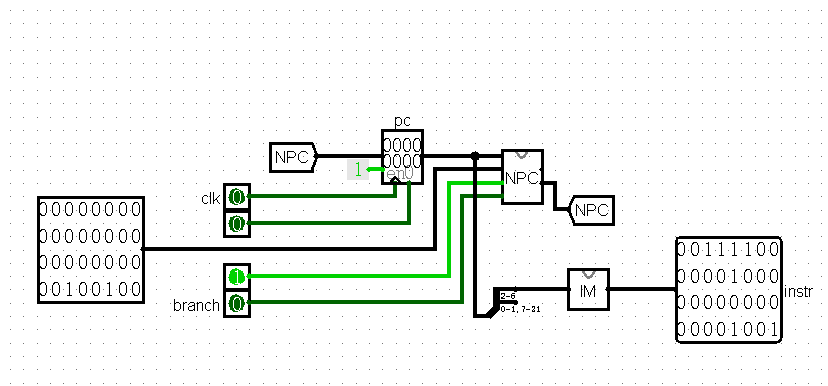
IFU接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| clk | I | 时钟信号 |
| Imm[31:0] | I | 32位立即数 |
| Zero | I | Alu传入的zero信号 |
| branch | I | branch信号，判断是否执行beq指令 |
| Re | I | 重置信号 |
| Instr[31:0] | O | 从IM中读取指令 |

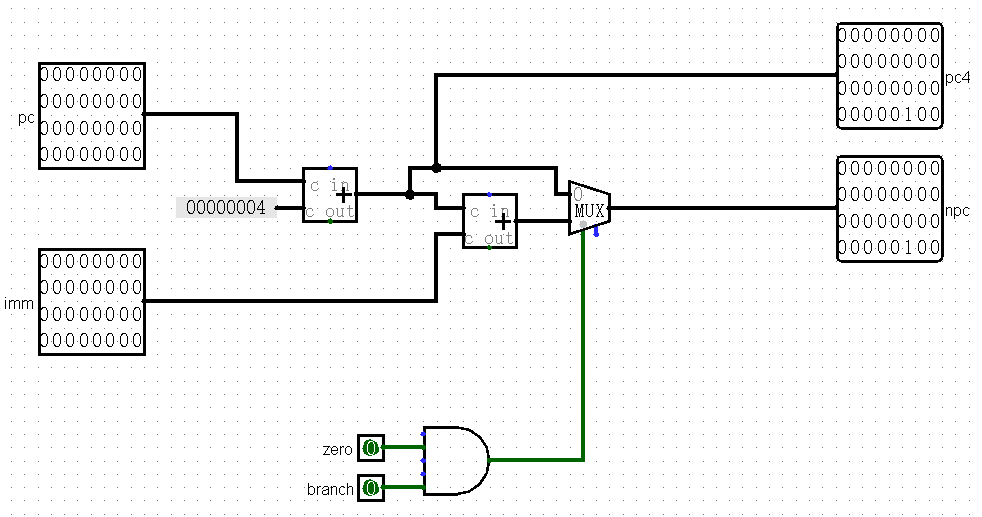
IFU功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算地址 | 计算pc下一个地址 |
| 2 | 读取指令 | 读取IM中存储的指令 |

图为IFU模块



图为IFU中NPC模块



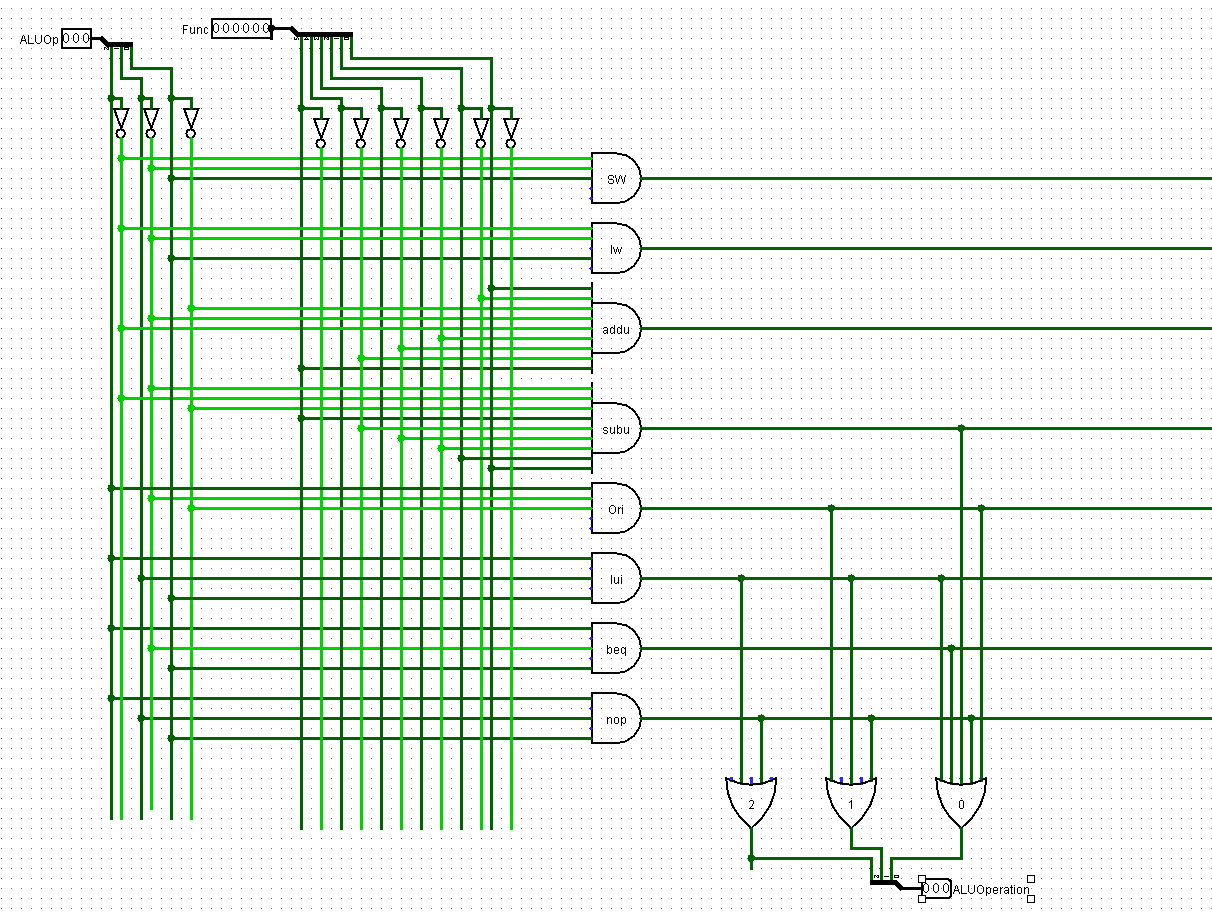
7.ALUControl

ALUControl接口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| ALUOp[2:0] | I | 输入3位控制信号 |
| Func[2:0] | I | 输入6位func信号 |
| ALUOperation[2:0] | O | 计算出ALUOperation的3位信号 |

ALUControl功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算ALUOperation值 | 结合ALUOp和Func计算出ALUOperation 的值 |



7.Control

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| Op[5:0] | I | Op信号 |
| RegDst | O | 写地址控制 选择RT,RD |
| Branch | O | 判断是否为beq指令，是则设置为1 |
| MemtoReg | O | GRF写入的选择信号 |
| MemWrite | O | DM写入信号 |
| MemRead | O | DM读入信号 |
| ALUOp[2:0] | O | 传递给ALUcontrol的控制信号 |
| ALUsrc | O | ALU第二操作数的选择信号 |
| RegWrite | O | GRF写入控制信号 |
| ExtOp | O | 控制 Ext有符号/无符号扩展信号 |

三、控制模块设计

1.ALUControl设计思路

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ALUOp | Func | 功能 | ALUOperation |
| Sw | 001 | xxxxxx | 加法 | 000 |
| lw | 001 | xxxxxx | 加法 | 000 |
| Addu | 000 | 100001 | 加法 | 000 |
| subu | 000 | 100011 | 减法 | 001 |
| Ori | 011 | xxxxxx | 或 | 011 |
| lui | 111 | xxxxxx | 无 | 111 |
| beq | 010 | xxxxxx | 减法 | 001 |
| nop | 111 | xxxxxx | 无 | xxx |

2.Control设计思路

1.对RegDst：1)R型指令，GRF的WR接口选择Rd，所以RegDst=1

2) lw,lui,ori指令，WR接口选择Rt， 故 RegDst = 0，其余时候都可以

2.branch信号：只有指令操作为beq时，为1；其余时候为0

3.MemtoReg，当指令为addu/subu时候，向GRF传入ALU 的值，MemtoReg=00。

当指令为lw,向GRF传入DM的值，MemtoReg = 01。

当指令为lui,向GRF传入立即数，MemtoReg=10，其余时候不做要求

4.MemWrite：只有sw时，需要向DM内写入值，MemWrite = 1

其余时候，MemWrite = 0.

5.MemRead：当指令为lw 时候，需要读入DM的值，MemRead =1

其余时候，MemRead = 0

6.ExtOp： 当指令为ori时候，需要进行符号扩展，ExtOp =0，进行符号扩展

其余时候：ExtOp = 1，进行无符号扩展

7.RegWrite：当指令为addu/subu/lw/loi/lui时，需要向GRF写入数据，此时RegWrite为1，其余时候为0

8.AluOp：当指令为addu/subu时候，ALUOp为000

当指令为lw/sw需要用到alu加法计算地址时，ALUOp为001

当指令为beq需要alu进行减法运算时，ALUOp为010

当指令为ori需要ALU进行或运算时，ALUOp为011

9.ALUSrc:当指令为lw/sw/ori需要涉及立即数运算时，ALUSrc为1，其余时候为0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | lw | sw | beq | lui | ori | nop |
| Op | 000000 | 000000 | 100011 | 101011 | 000100 | 001111 | 001101 | 000000 |
| RegDst | 1 | 1 | 0 | x | x | 0 | 0 | x |
| branch | 0 | 0 | 0 | 0 | 1 | 0 | 0 | x |
| MemtoReg | 00 | 00 | 01 | xx | xx | 10 | 00 | x |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 | 0 | x |
| MemRead | 0 | 0 | 1 | 0 | 0 | 0 | 0 | x |
| ALUOp | 000 | 000 | 001 | 001 | 010 | 111 | 011 | 111 |
| ALUSrc | 0 | 0 | 1 | 1 | 0 | x | 1 | x |
| RegWrite | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x |
| ExtOp | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x |

四、测试程序设计

.text

lui $t0, 100 #t0 高位赋值100

ori $t1, $t0, 234 #T1 = t0 | 234

addu $t2, $t1, $t0 # t2 = t1 + t0

addu $t3, $t2, $t2 # t3 = t2 + t2

subu $t4, $t3, $t0 #t4 = t3 - t0

lui $a0, 100 #a0 高位赋值100

lui $a1, 100

lui $v0, 400

beq $v0, $a0, end1 #若正常运行，则此处不应跳转

sw $v0, 4($0)#向0x4存入v0的值

sw $v0, 8($0)

sw $a1,12($0)

lw $v1, 4($0)#从0x00000004取出v1的值

lw $s5, 8($0)

nop

beq $a0, $a1, end #若正常，则s0不应被赋值

lw $s0,4($a0)

end1:

end:

lui $v0, 200

16进制机械码：

v2.0 raw

3c080064

350900ea

01285021

014a5821

01686023

3c040064

3c050064

3c020190

10440008

ac020004

ac020008

ac05000c

8c030004

8c150008

00000000

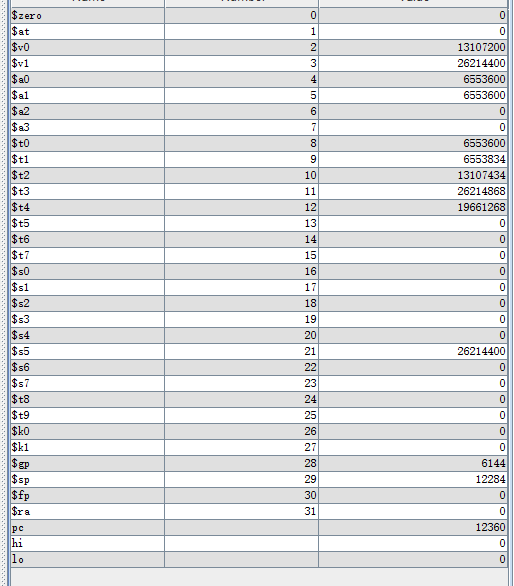
10850001

8c900004

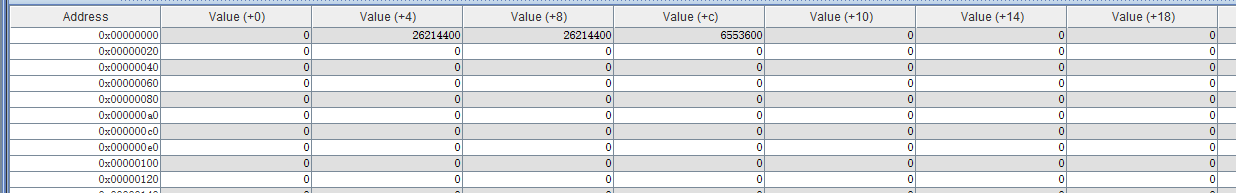
3c0200c8

Mars运行后结果：

寄存器区：

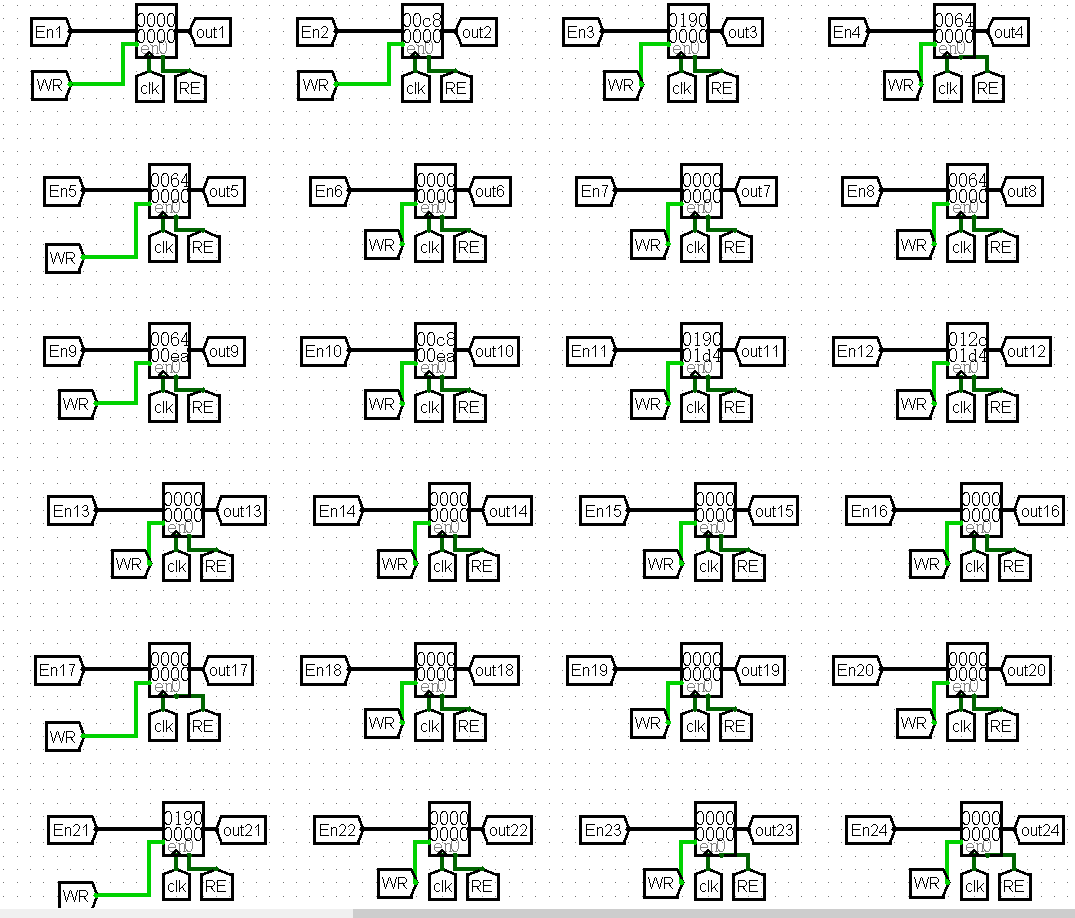


内存区：

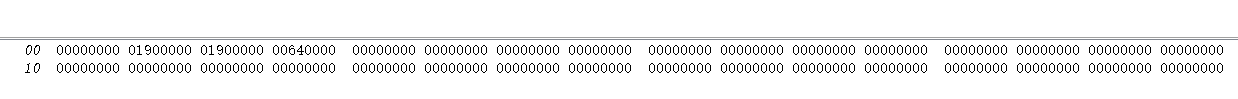


Logisim运行后的结果

寄存器区：



内存区：



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：1.30位pc会使存储容量变小四倍，2.执行pc = pc + 4时候，进行拓展。

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。

我认为合理，ROM为只读存储器，RAM为内存中存储器，可读可写，力求范围大，对速度要求不高，RAM比较适合。最后GRF存储，对范围要求少，对速度要求高，所以适合用寄存器

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：将Op，func分为Op5,Op4,Op3,Op2,Op1,Op0和f5,f4,f3,f2,f1,f0

RegDst = !Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0

ALUSrc = (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

MemtoReg = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

RegWrite = (!Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0) | (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

MemWrite = (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

nPC\_sel = !Op5 & !Op4 & !Op3 & Op2 & !Op1 & !Op0

ExtOp = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

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。

即，为了简化可将X取为0/1，在本题中将X均取为0，化简后为:

RegDst = !Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0

ALUSrc = (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

MemtoReg = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

RegWrite = (!Op5 & !Op4 &!Op3 &!Op2 & !Op1 & !Op0 & f5 & !f4 & !f3 & !f2 & !f0) | (!Op5 & !Op4 & Op3 & Op2 &!Op1 & Op0) | (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0)

MemWrite = (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

nPC\_sel = !Op5 & !Op4 & !Op3 & Op2 & !Op1 & !Op0

ExtOp = (Op5 & !Op4 & !Op3 & !Op2 & Op1 & Op0) | (Op5 & !Op4 & Op3 & !Op2 & Op1 & Op0)

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 的机械码为0X00000000，即什么都不执行，只执行pc = pc + 4操作，所以不需要加入真值表，因为对任何控制信号都无需求。

4. 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。

5. 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。

优点：

1.形式验证的覆盖率达到100%，对所有可能出现的情况进行模拟

2.形式验证是利用数学上的方法，将待验证电路和功能描述进行比较，不用仿真模拟

3.形式验证是进行逻辑形式的比较，由于逻辑结构区别不大，因此验证时间短，可以快速发现错误

缺点：

不能有效地测试电路的性能，如功耗等。