Project3 单周期CPU开发

## CPU设计方案综述

1. 总体设计概述
2. 处理器为32为处理器。
3. 支持的指令集为{addu,lui,subu,ori,beq,nop,lw,sw}.
4. 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）。
5. addu，subu 可以不支持溢出。
6. 处理器为单周期设计。
7. 关键模块定义
8. IFU

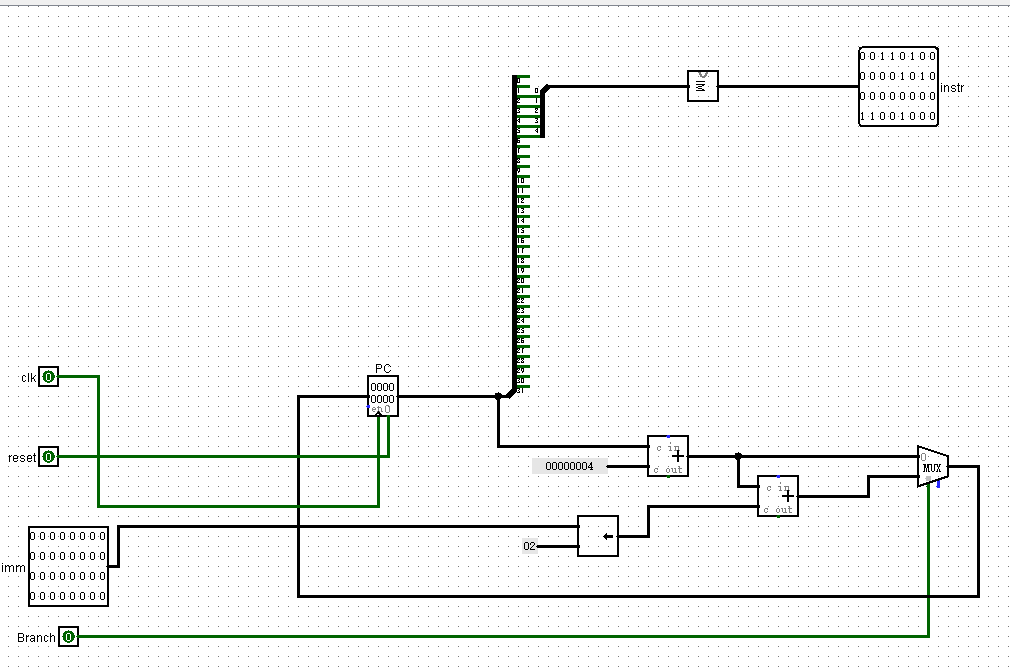
介绍：内部包括PC（程序计数器）、IM（指令存储器）及相关逻辑，PC用寄存器实现，具有异步复位功能，复位值为起始地址，IM用ROM实现，容量为32bit \* 32。

端口定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 功能描述 |
| Clk | I | 1 | 时钟信号 |
| reset | 1 | 1 | 复位信号 |
| imm | I | 32 | beq指令的sign\_extend(offset ||02 ) 32位立即数 |
| Branch | I | 1 | 跳转信号 |
| instr | O | 32 | 取出来的指令 |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 如果reset有效，则进行复位，复位PC为0x00000000 |
| 2 | 取指令 | PC从IFU取出相应的指令 |
| 3 | 跳转 | 当Branch有效时，PC = PC + 4 + imm |



1. GRF

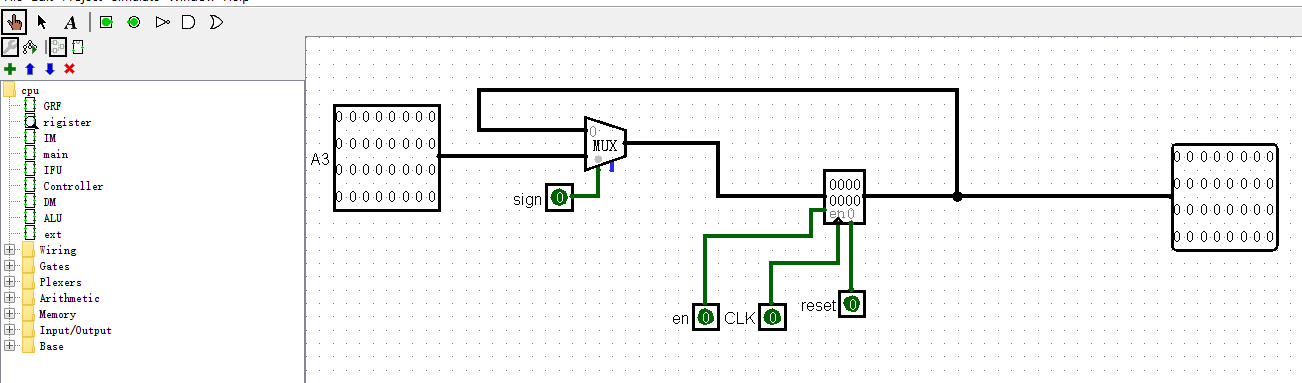
介绍：用具有写使能的寄存器实现，寄存器总数为32个，应具有异步复位功能，其中0号寄存器的值始终为0。

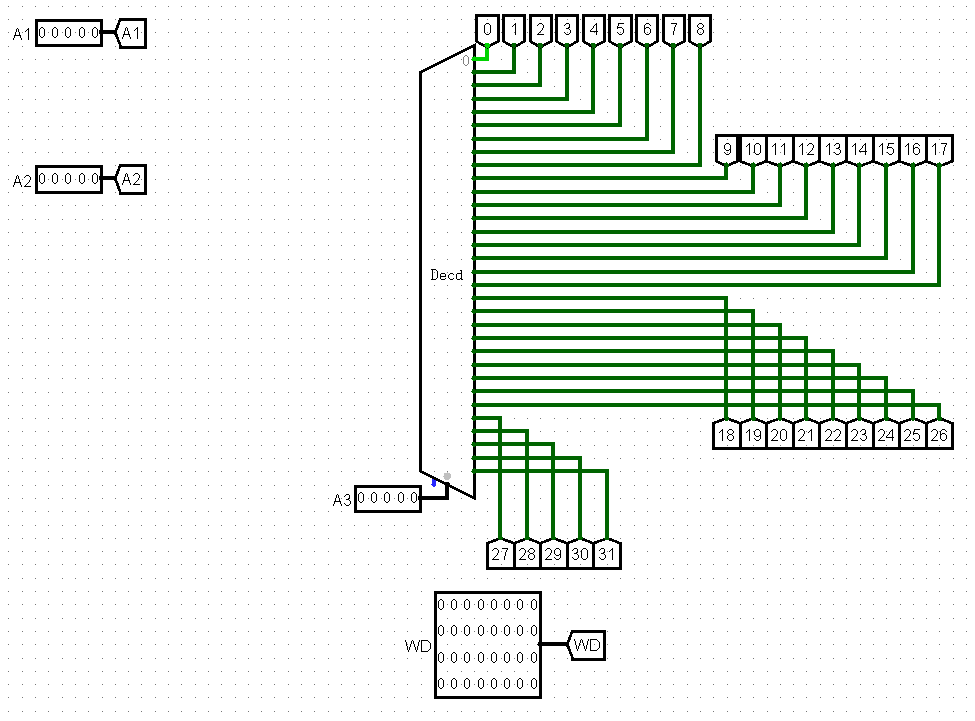
端口定义：

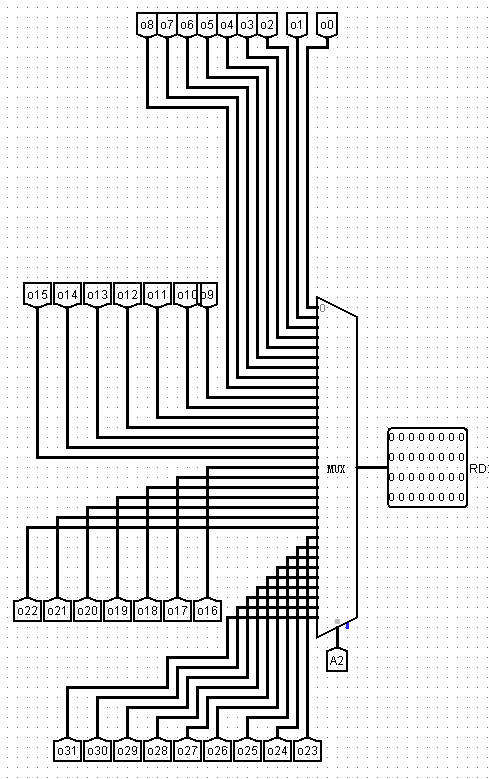
|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 描述 |
| clk | I | 1 | 时钟信号 |
| reset | I | 1 | 复位信号，将32个寄存器中的值全部清零 1:复位 0：无效 |
| WE | I | 1 | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| A1 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1 |
| A2 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2 |
| A3 | I | 5 | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| WD | I | 32 | 32位数据输入信号 |
| RD1 | I | 32 | 输出A1指定的寄存器中的32位数据 |
| RD2 | I | 32 | 输出A2指定的寄存器中的32位数据 |

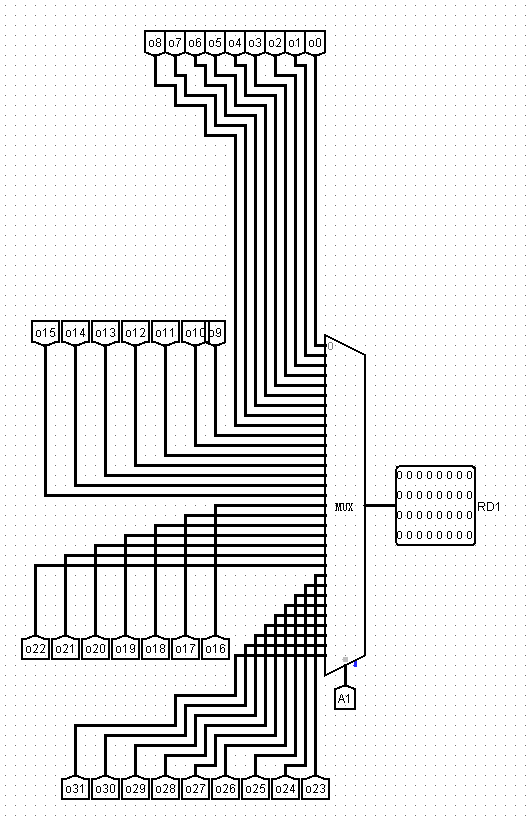
功能定义：

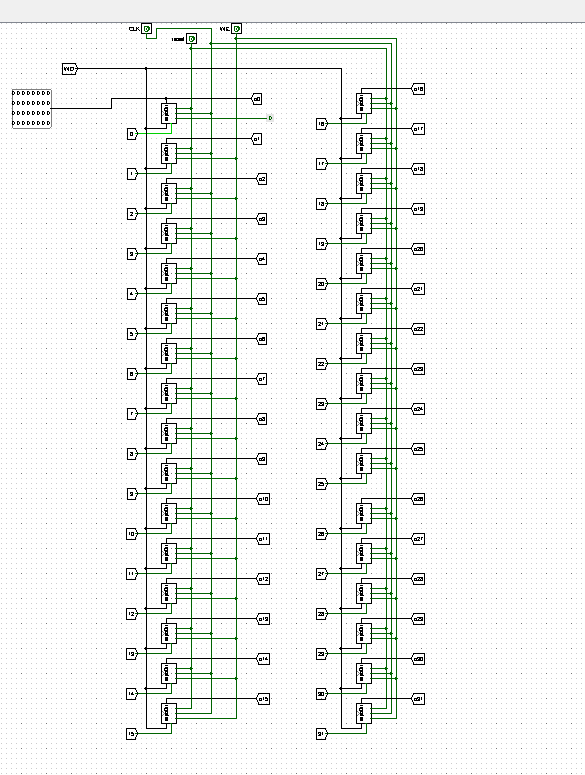
|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | Reset信号有效时，所有寄存器中存储的数值清零，其行为与logisim自带部件register的reset接口完全相同 |
| 2 | 读数据 | 读出A1、A2地址对应寄存器中所存储的数据到RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |











1. ALU

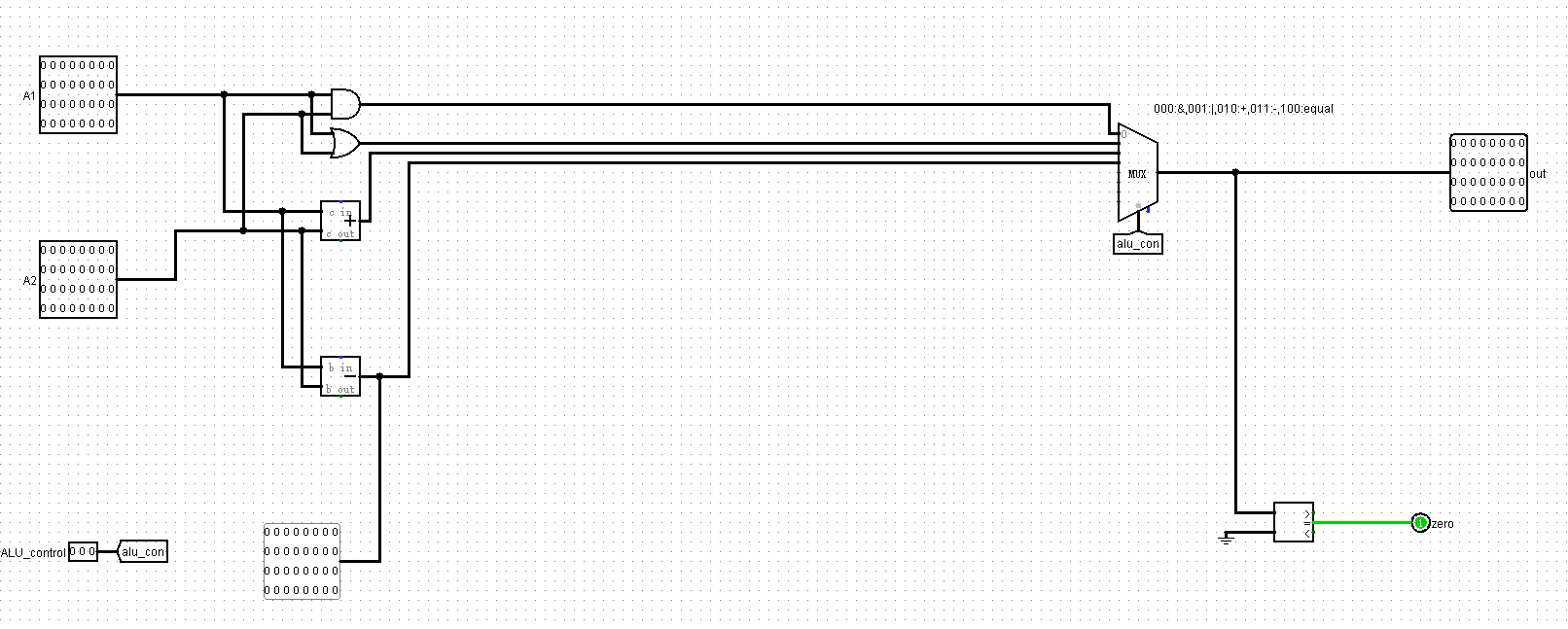
介绍：提供32位加、减、或运算以及大小比较功能，不检测溢出。

端口定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 描述 |
| A1 | I | 32 | 操作数1 |
| A2 | I | 32 | 操作数2 |
| ALU\_control | I | 32 | ALU运算控制信号 |
| out | O | 32 | 操作结果 |
| zero | O | 32 | 判断结果是否为0 |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 或运算 | 当ALU\_control=001时，进行或运算 |
| 2 | 加法运算 | 当ALU\_control=010时，进行加法运算 |
| 3 | 减法运算 | 当ALU\_control=011时，进行减法运算 |
| 4 | 判断两数是否相等 | 当ALU\_control=011时，进行减法运算，若计算结果为0，则zero为1，说明两数相等，否则说明两数不相等 |



1. DM

介绍：使用RAM实现，容量为32bit \* 32，应具有异步复位功能，复位值为0x00000000。

端口定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 描述 |
| clk | I | 1 | 时钟信号 |
| reset | I | 1 | 复位信号 |
| WE | I | 1 | 写入控制信号  WE=1时，写入  WE=0时，不写入 |
| MemaRead | I | 1 | 读出控制信号  MemRead=1时读出；  MemRead=0时不读出 |
| A | I | 5 | 需要读出或写入数据的地址 |
| WD | I | 32 | 写入的数据 |
| RD | I | 32 | 读出的数据 |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset=1时，将DM中的数据清零 |
| 2 | 写入数据 | 当WE=1时，将WD写入A地址中 |
| 3 | 读数据 | 当MemRead=1时，将A地址对应的数据读出到RD中 |

1. EXT

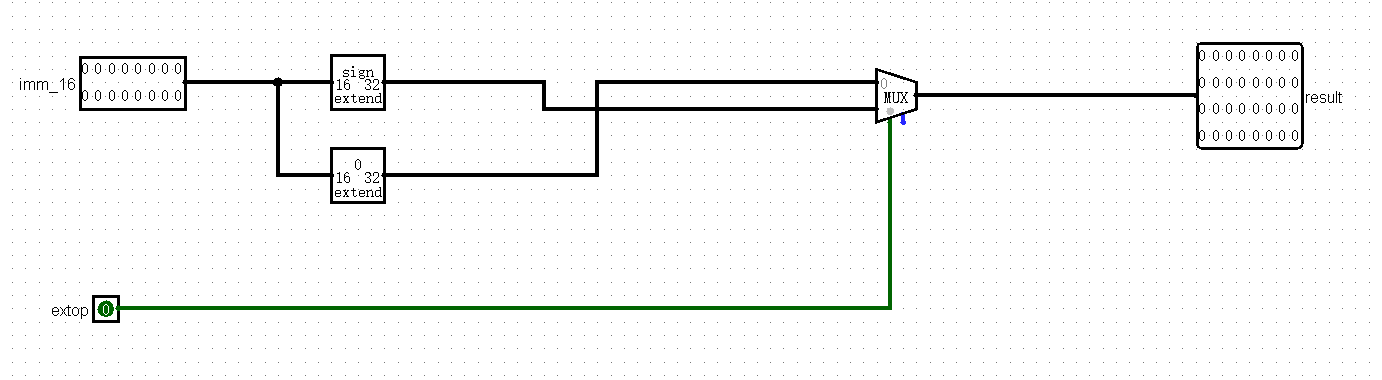
介绍：支持16bit到32bit的零拓展以及符号拓展

端口定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 描述 |
| Imm\_16 | I | 16 | 需要拓展的16bit立即数 |
| Extop | I | 1 | 是否进行符号拓展  1：符号拓展  0：零拓展 |
| Result | O | 32 | 拓展结果 |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零拓展 | 当extop=0时，进行零拓展 |
| 2 | 符号拓展 | 当extop=1时，进行符号拓展 |



1. Controller

介绍：使用与或门阵列构造控制信号。

端口定义：

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 方向 | 位宽 | 描述 |
| Opcode | I | 6 | Instr[31:26] |
| Func | I | 6 | Instr[5:0] |
| RegRead\_2 | O | 1 | 控制是否读入第二个寄存器的地址 |
| RegWrite | O | 1 | 寄存器写入控制信号 |
| Extop | O | 1 | 是否进行符号拓展 |
| MemWrite | O | 1 | 内存写入控制信号 |
| RegRead\_1 | O | 1 | 判断是否读入第一个寄存器的地址 |
| Alu\_2 | O | 1 | 判断参与ALU运算的第二个操作数是否是立即数 |
| MemRead | O | 1 | 内存读出控制信号 |
| Branch | O | 1 | 跳转信号 |
| Alu\_control | O | 3 | 运算控制信号 |
| RegDst | O | 1 | 写入的寄存器选择信号 |
| sw | O | 1 | 判断指令是否是sw |

功能定义：

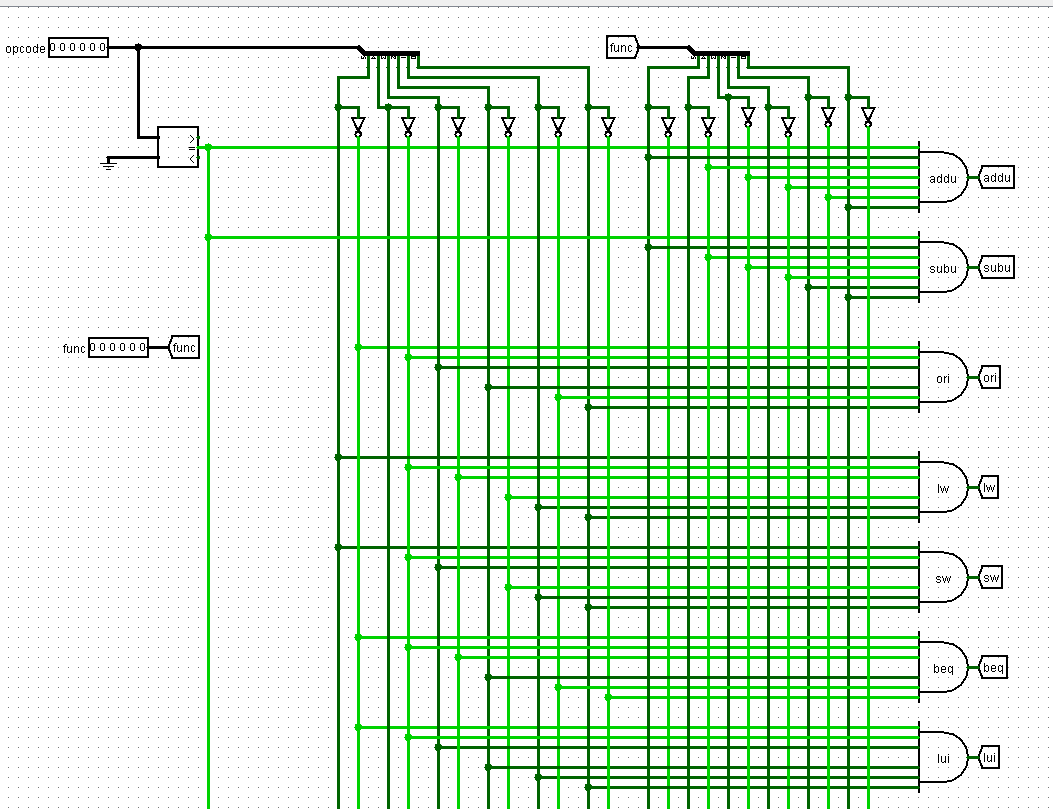
|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生控制信号 | 产生控制信号 |

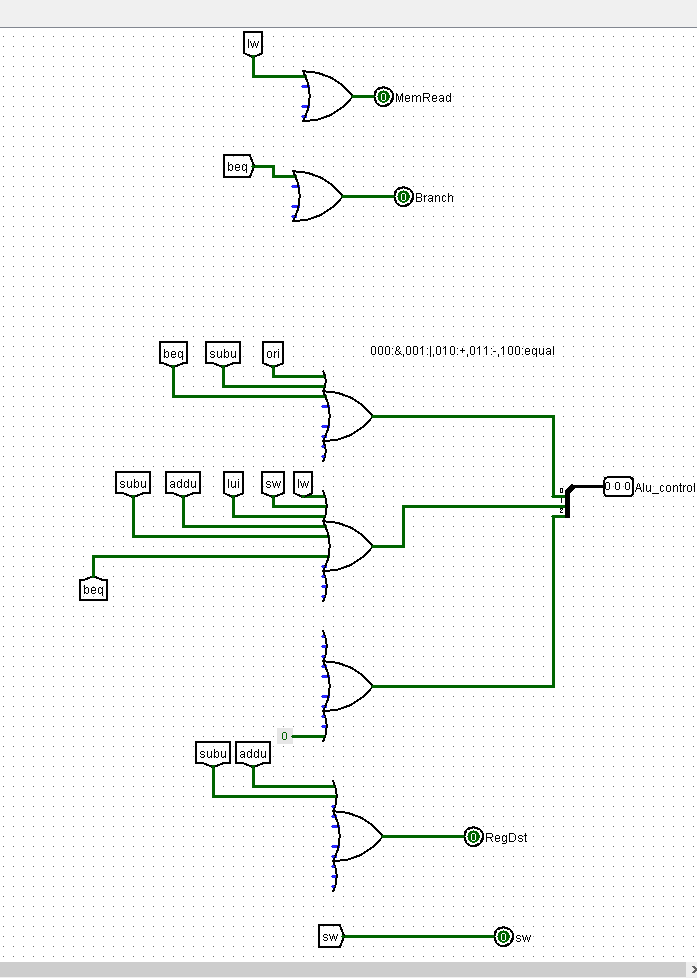
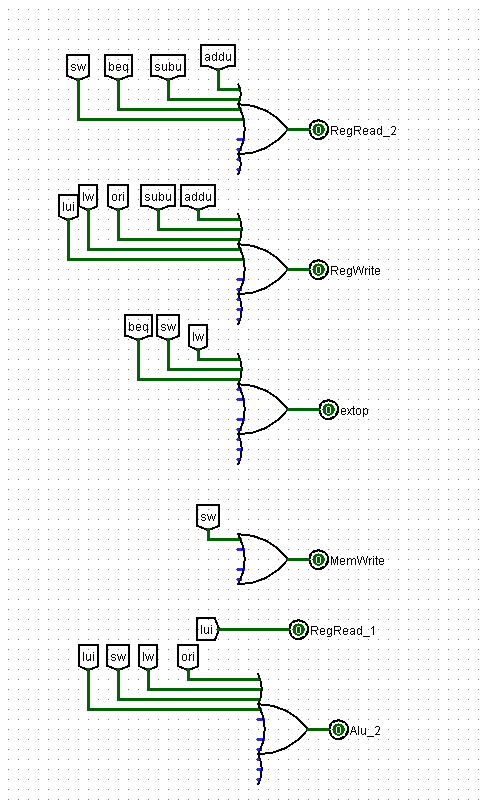
1. 重要机制实现方法

控制信号的产生：运用与或门阵列生成

**控制信号真值表**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | ori | beq | Lw | sw |  | lui | nop |
| opcode | 000000 | 000000 | 001101 | 000100 | 100011 | 101011 |  | 001111 | 000000 |
| Func | 100001 | 100011 |  |  |  |  |  |  | 000000 |
| RegRead\_2 | 1 | 1 |  | 1 |  | 1 |  |  |  |
| RegWrite | 1 | 1 | 1 |  | 1 |  |  | 1 |  |
| extop |  |  |  | 1 | 1 | 1 |  |  |  |
| MemWrite |  |  |  |  |  | 1 |  |  |  |
| RegRead\_1 |  |  |  |  |  |  |  | 1 |  |
| Alu\_2 |  |  | 1 |  | 1 | 1 |  | 1 |  |
| MemRead |  |  |  |  | 1 |  |  |  |  |
| Branch |  |  |  | 1 |  |  |  |  |  |
| Alu\_control | 010 | 011 | 001 | 011 | 010 | 010 |  | 010 |  |
| RegDst | 1 | 1 |  |  |  |  |  |  |  |
| sw |  |  |  |  |  | 1 |  |  |  |





## 二、测试方案

典型测试样例

测试ori指令

ori $0, $0, 0

ori $1, $1, 1

ori $2, $2, 2

ori $3, $3, 3

ori $4, $4, 4

ori $5, $5, 5

ori $6, $6, 6

ori $7, $7, 7

ori $8, $8, 8

ori $9, $9, 9

ori $10, $10, 10

ori $11, $11, 11

ori $12, $12, 12

ori $13, $13, 13

ori $14, $14, 14

ori $15, $15, 15

ori $16, $16, 16

ori $17, $17, 17

ori $18, $18, 18

ori $19, $19, 19

ori $20, $20, 20

ori $21, $21, 21

ori $22, $22, 22

ori $23, $23, 23

ori $24, $24, 24

测试sw指令

ori $1, $1, 20

sw $1, 4($0)

sw $1, 8($0)

sw $1, 12($0)

sw $1, 16($0)

sw $1, 20($0)

sw $1, 24($0)

sw $1, 28($0)

sw $1, 32($0)

sw $1, 36($0)

sw $1, 40($0)

sw $1, 44($0)

sw $1, 48($0)

sw $1, 52($0)

sw $1, 56($0)

sw $1, 60($0)

sw $1, 64($0)

sw $1, 68($0)

sw $1, 72($0)

测试lw指令

ori $1, $1, 1

sw $1, 0($0)

lw $2, 0($0)

lw $3, 0($0)

lw $4, 0($0)

lw $5, 0($0)

lw $6, 0($0)

lw $7, 0($0)

lw $8, 0($0)

lw $9, 0($0)

lw $10, 0($0)

lw $11, 0($0)

lw $12, 0($0)

lw $13, 0($0)

lw $14, 0($0)

lw $15, 0($0)

lw $16, 0($0)

lw $17, 0($0)

lw $18, 0($0)

lw $19, 0($0)

测试lui指令

lui $0，0

lui $1,1

lui $2,2

lui $3,3

测试addu，subu，beq，nop

ori $1, $1, 10

ori $2, $2, 20

addu $3, $1, $1

addu $4, $1, $2

subu $5, $2, $1

subu $6, $1, $1

lui $4, 0xffff

nop

beq $1, $0, end

lui $1, 0xffff

end:

## 三、思考题

1.现在我们的模块中IM使用ROM， DM使用RAM， GRF使用Register，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。IM在数据导入之后无需再进行写入操作，只有取出操作，因此选用ROM很合理；DM可读可写，且需要大量内存，ROM不仅容量大，且可读可写，所以很合理；GRF要求读写的速度要快，而rigister的速度是这三者中最快的，所以GRF用ROM正好合适。

2.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？

Nop指令是0x00000000，对于整个电路而言，只是做了pc + 4，并没有对电路的其它原件产生影响，因此没必要将其加入控制信号。

3.上文提到，MARS 不能导出 PC 与 DM 起始地址均为 00 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦。请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

我们可以直接减去地址的偏移量，将DM整体减去一个0x30000000。

4.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，简要阐述相比于测试，形式验证的优劣之处。

优点：形式验证是使用数学方法对所有可能的情况进行验证，而通过测试无法枚举所有的情况，因此形式验证更为完备和严谨。

缺点：由于形式验证需要借助数学工具，相比于直接测试操作更为复杂，耗费的经历也越多。