Project3 单周期 CPU 开发

一、CPU 设计方案综述

（一） 总体设计概述

1. 处理器为 32 为处理器。
2. 支持的指令集为{addu, lui, subu, ori, beq, nop, lw, sw}.
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。

（二） 关键模块定义

1) 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。

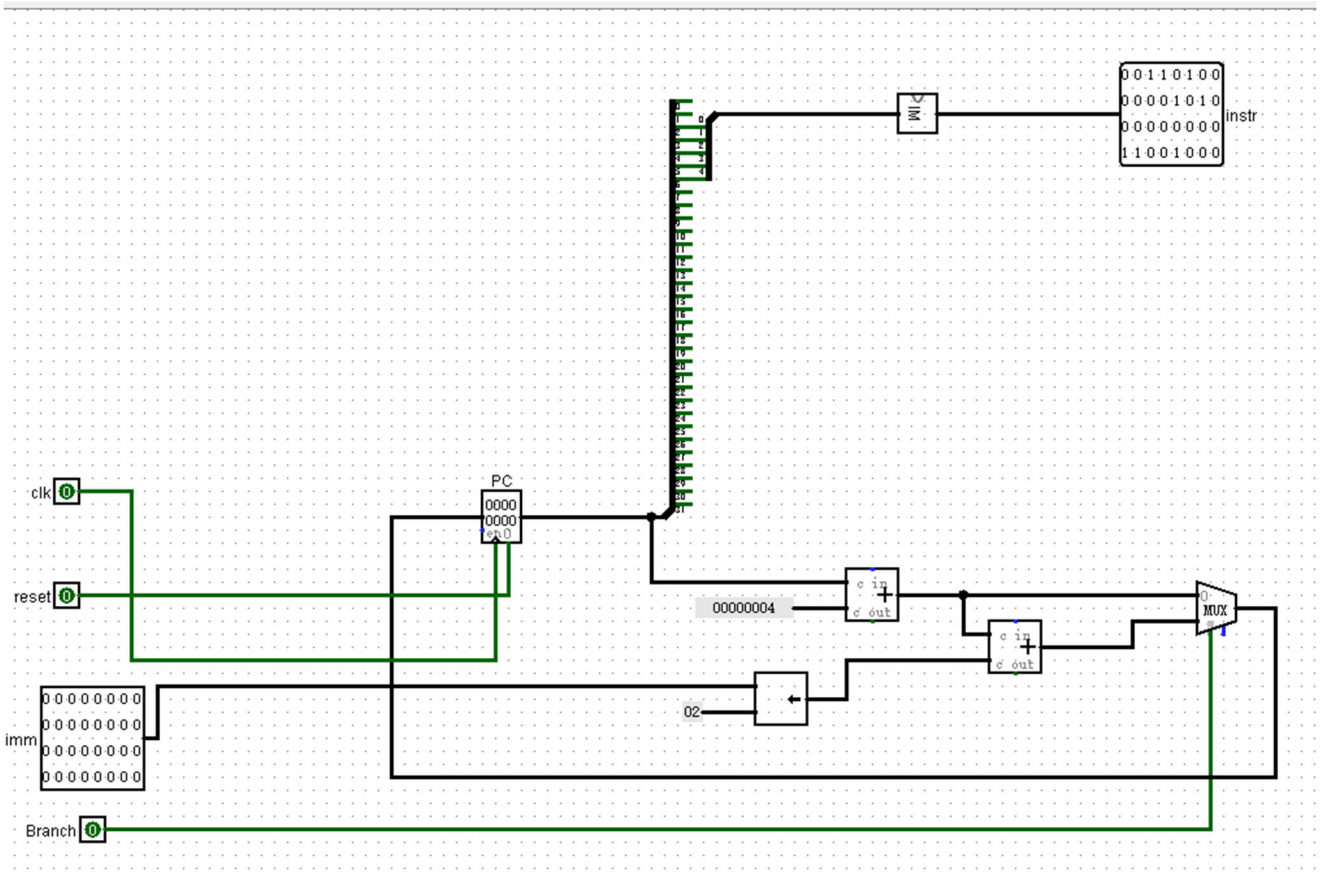
端口定义：

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 ||0^2）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



2) 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。

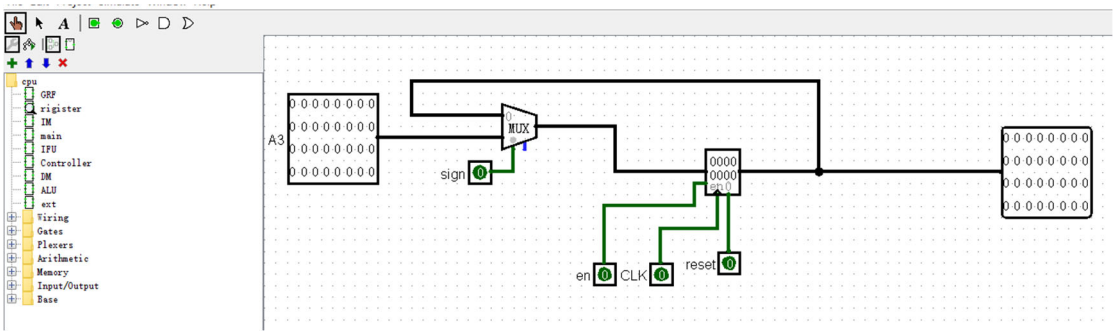
端口定义：

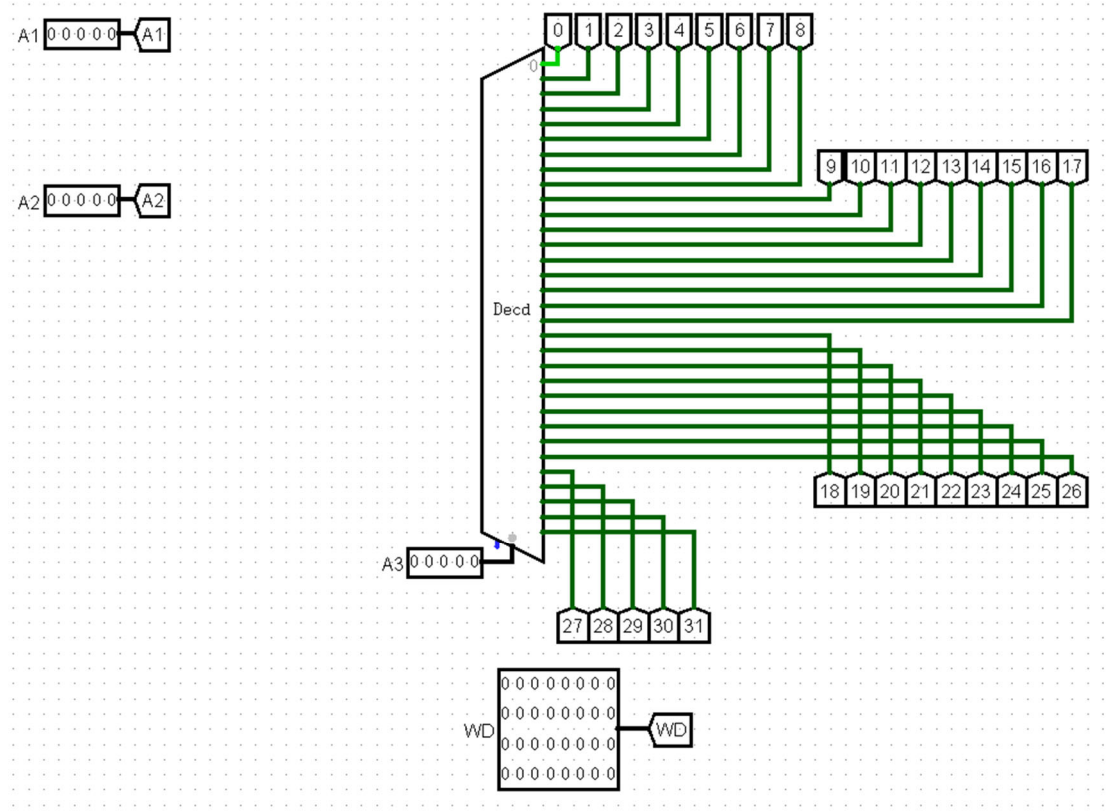
端口	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	复位信号，将 32 个寄存器中的值全部清零 1:复位 0：无效
WE	I	1	写使能信号 1：可向 GRF 中写入数据 0：不能向 GRF 中写入数据
A1	I	5	5 位地址输入信号，指定 32 个寄

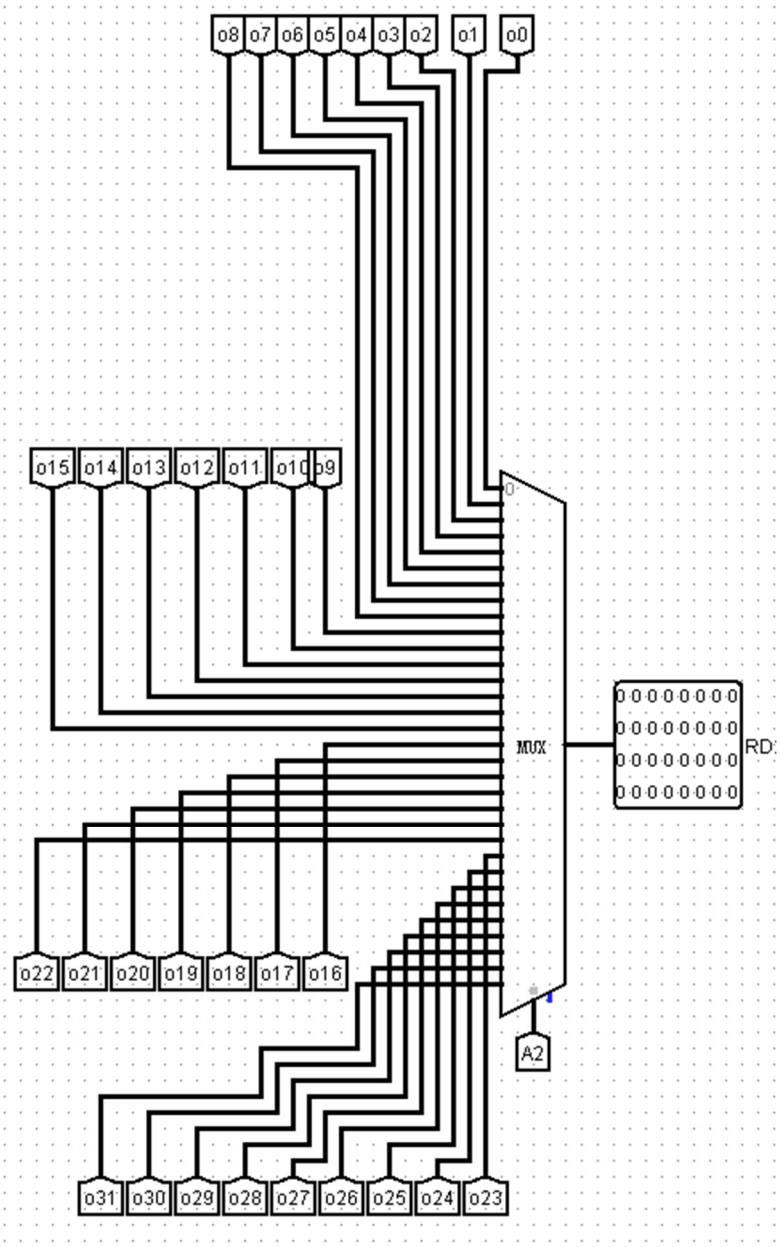
			存器中的一个，将其中存储的数据读出到 RD1
A2	I	5	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
A3	I	5	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WD	I	32	32 位数据输入信号
RD1	I	32	输出 A1 指定的寄存器中的 32 位数据
RD2	I	32	输出 A2 指定的寄存器中的 32 位数据

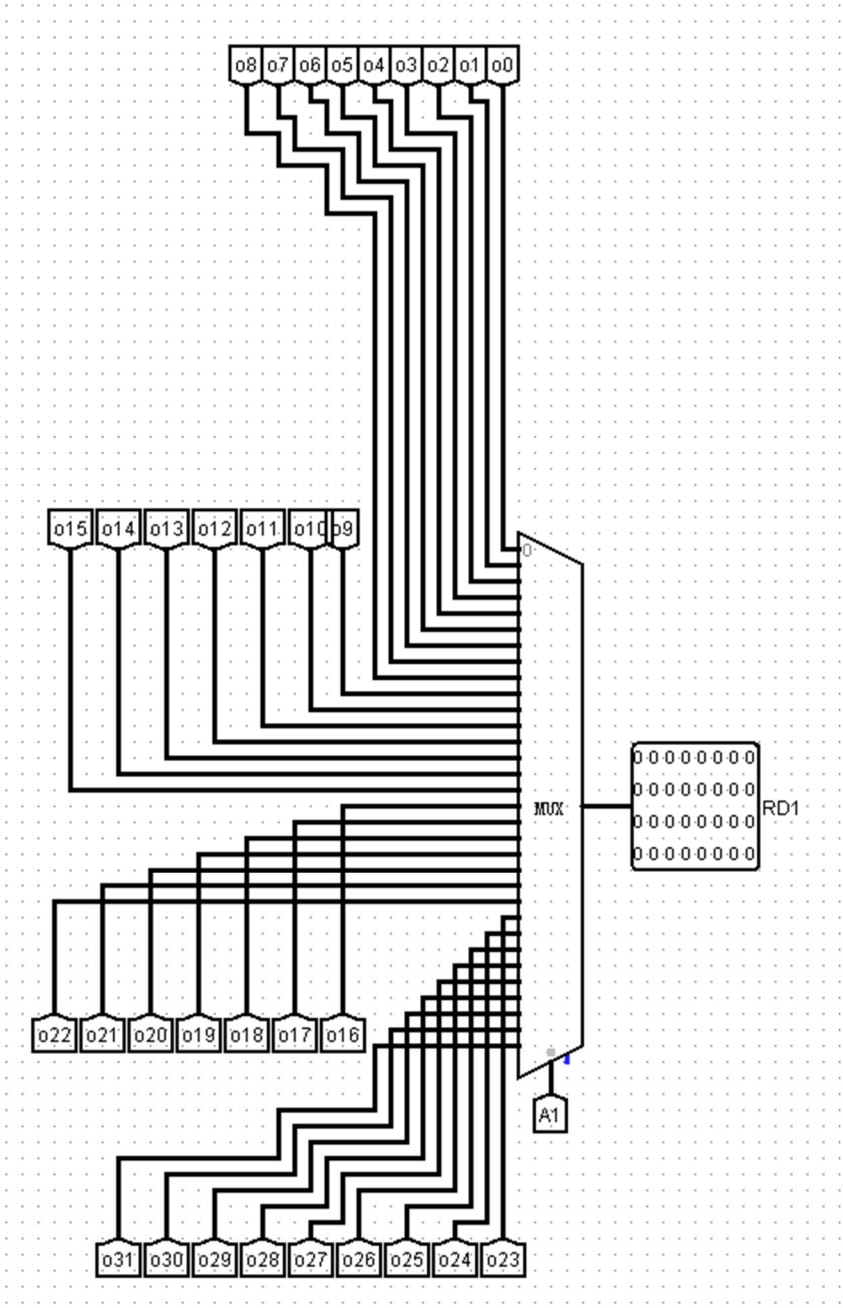
功能定义：

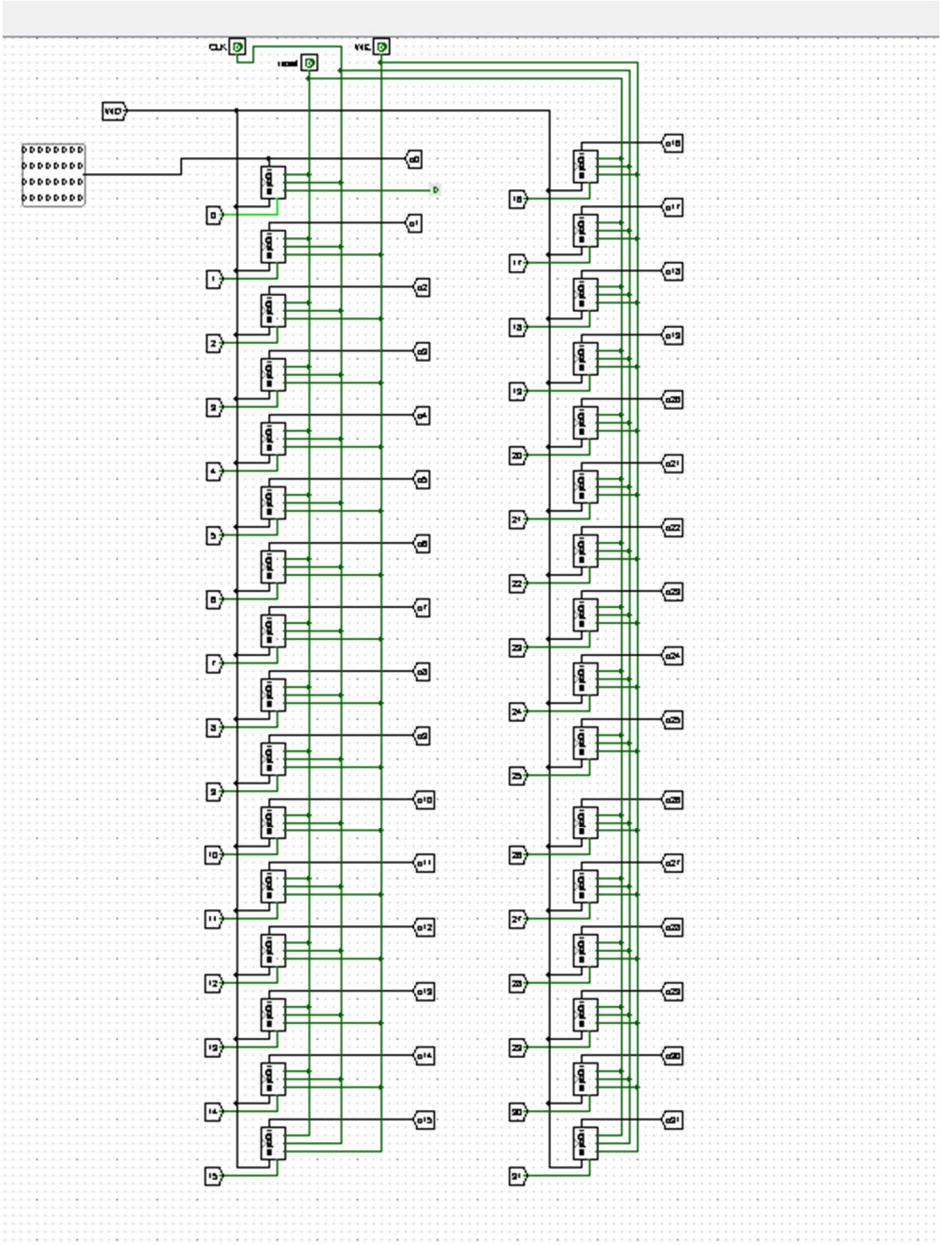
序号	功能名称	功能描述
1	复位	Reset 信号有效时，所有寄存器中存储的数值清零，其行为与 logisim 自带部件 register 的 reset 接口完全相同
2	读数据	读出 A1、A2 地址对应寄存器中所存储的数据到 RD2
3	写数据	当 WE 有效且时钟上升沿来临时，将 WD 写入 A3 所对应的寄存器中


cpu
GRF
rigister
IM
main
IFU
Controller
DM
ALU
ext
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
Base
A3
MUX
sign
en
CLK
reset


A1
A2
A3
Decd
WD
0 1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26
27 28 29 30 31


o8
o7
o6
o5
o4
o3
o2
o1
o0
o15
o14
o13
o12
o11
o10
o9
0
MUX
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
RD
o22
o21
o20
o19
o18
o17
o16
A2
o31
o30
o29
o28
o27
o26
o25
o24
o23


o8
o7
o6
o5
o4
o3
o2
o1
o0
o15
o14
o13
o12
o11
o10
o9
0
MUX
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
RD1
o22
o21
o20
o19
o18
o17
o16
A1
o31
o30
o29
o28
o27
o26
o25
o24
o23



3） 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。

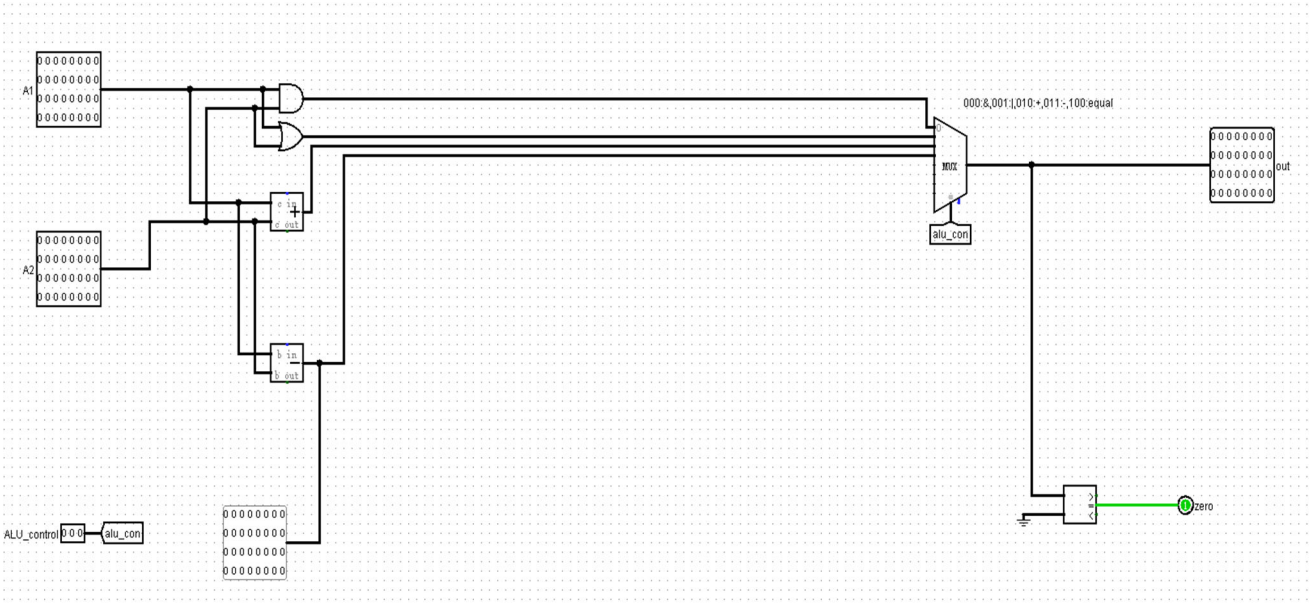
端口定义：

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



4） 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

5) EXT

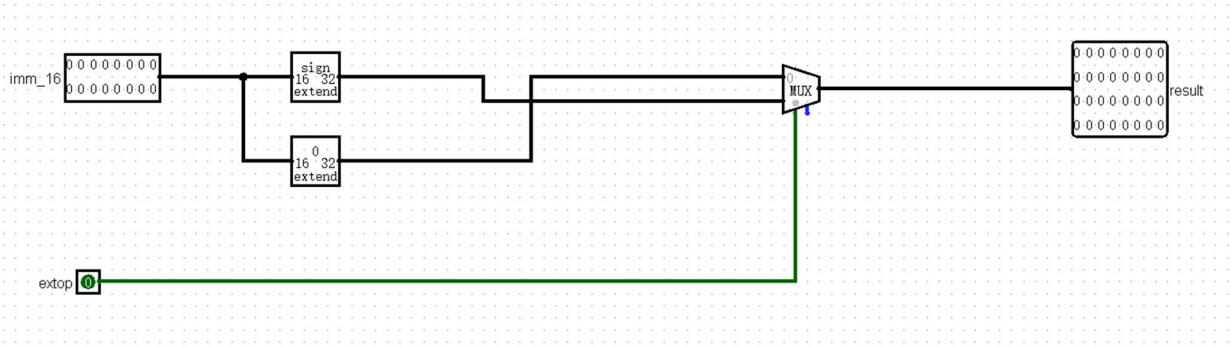
介绍：支持 16bit 到 32bit 的零拓展以及符号拓展

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



6) 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	0	1	控制是否读入第二个寄存器的地址
RegWrite	0	1	寄存器写入控制信号
Extop	0	1	是否进行符号拓展
MemWrite	0	1	内存写入控制信号
RegRead_1	0	1	判断是否读入第一个寄存器的地址
Alu_2	0	1	判断参与 ALU 运算的第二个操作数是否是立即数
MemRead	0	1	内存读出控制信号
Branch	0	1	跳转信号
Alu_control	0	3	运算控制信号
RegDst	0	1	写入的寄存器选择信号
sw	0	1	判断指令是否是 sw

功能定义：

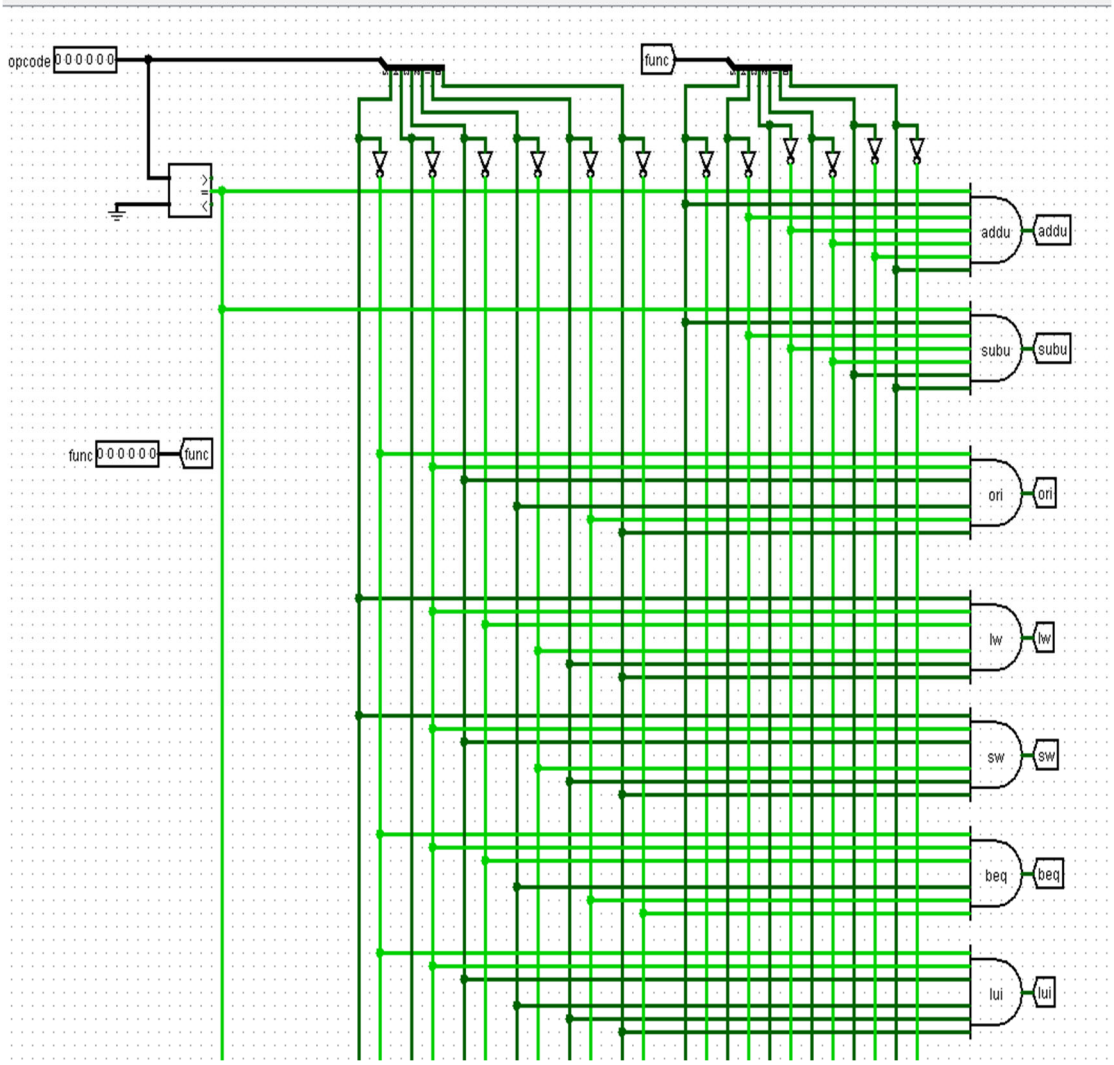
序号	功能名称	功能描述
1	产生控制信号	产生控制信号

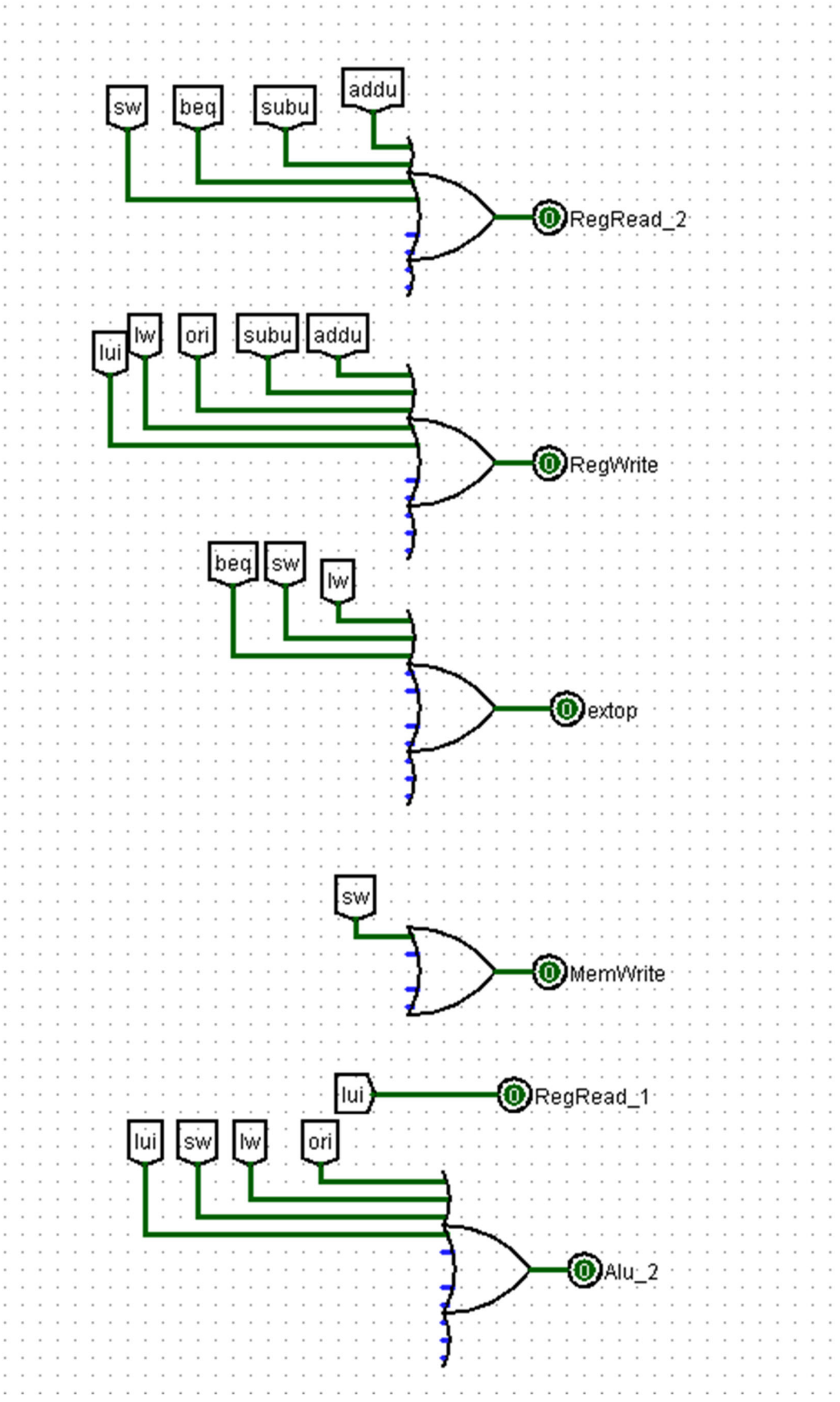
（三） 重要机制实现方法

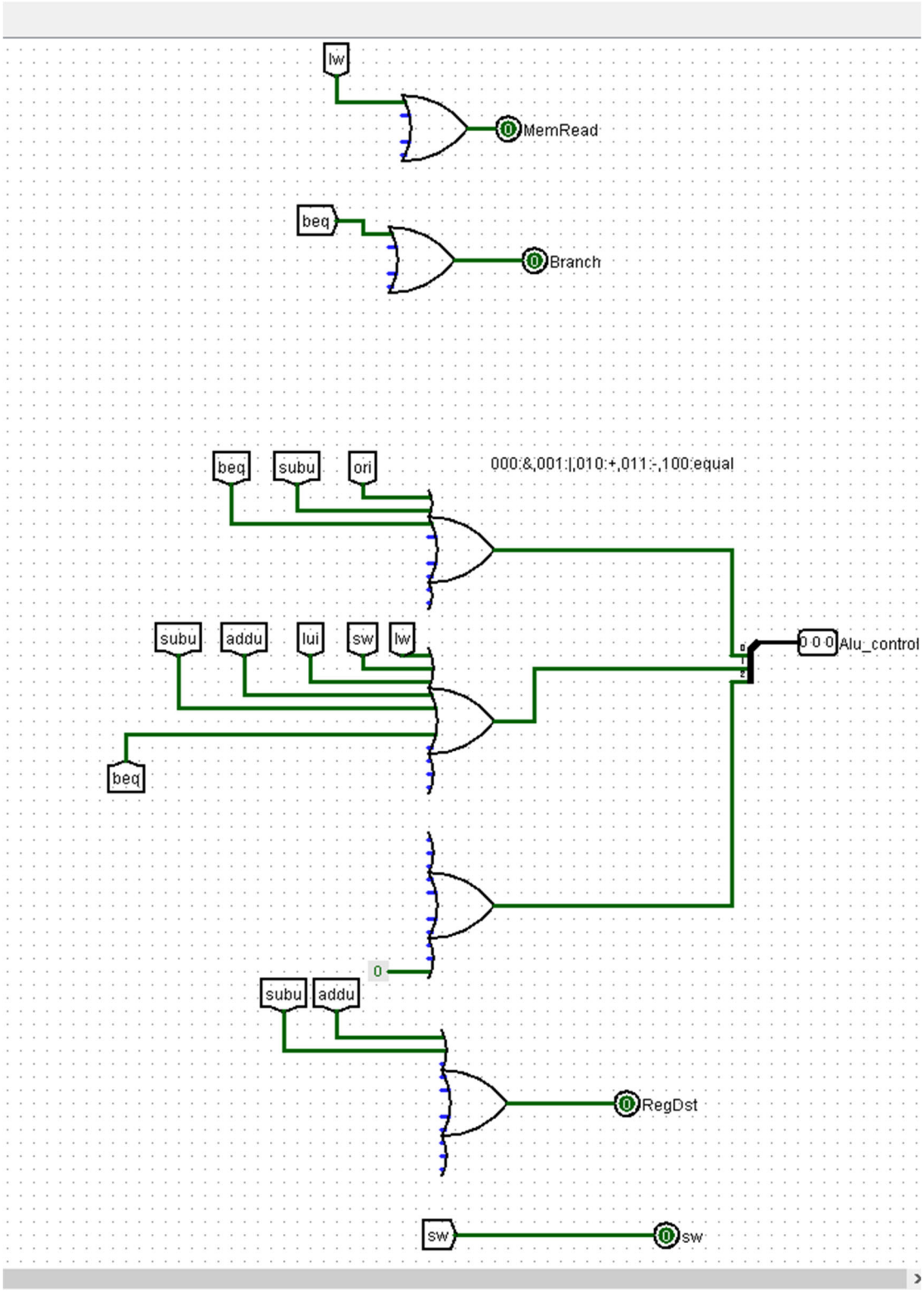
控制信号的产生：运用与或门阵列生成

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			


opcode 0 0 0 0 0 0
func 0 0 0 0 0 0
func
func
addu
addu
subu
subu
ori
ori
lw
lw
sw
sw
beq
beq
lui
lui


sw
beq
subu
addu
RegRead_2
lui
lw
ori
subu
addu
RegWrite
beq
sw
lw
extop
sw
MemWrite
lui
RegRead_1
lui
sw
lw
ori
Alu_2



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。