Project3 Logisim 完成单周期处理器开发实验报告

一．整体结构

**1.处理器为32位处理器。**

**2.处理器应支持的指令集为：{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。

6.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．模块规格

1.IFU

模块接口

|  |  |  |
| --- | --- | --- |
| 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 |
| Zero | I | ALU 计算结果是否为0的标志信号  1：计算结果为 0  0：计算结果非 0 |
| Branch | I | 当前指令是否为beq指令  1：是beq  0:不是beq |
| Instr | O | 32位MIPS指令 做输出 |
| Imm32 | I | beq指令的sign\_extend(offset ||02 ) 32位立即数 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 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+4  如果当前指令是 beq 指令，且Zero=1，则PC=PC+4+ sign\_extend(offset ||02) |

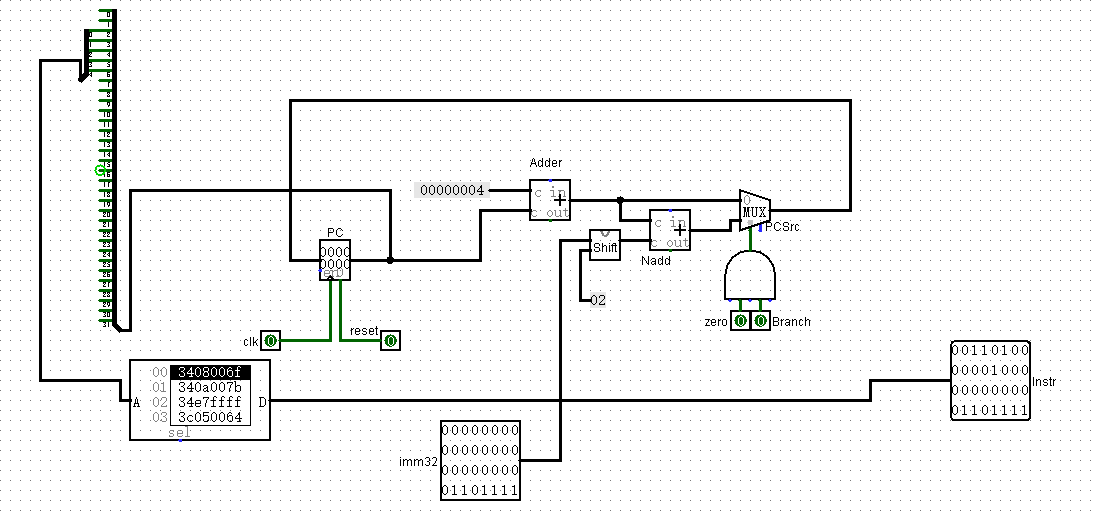
IM

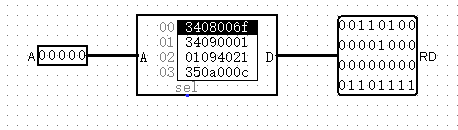
模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| A[4:0] | I | 5位地址 |
| RD[31:0] | O | 32位指令 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指令 | 根据 PC 从 IM 中取出指令 |





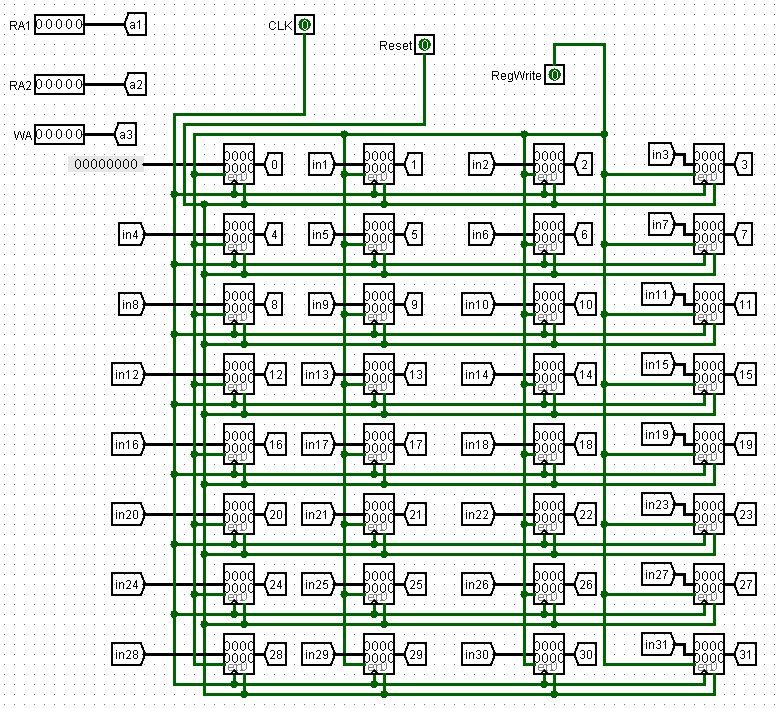
2.GRF

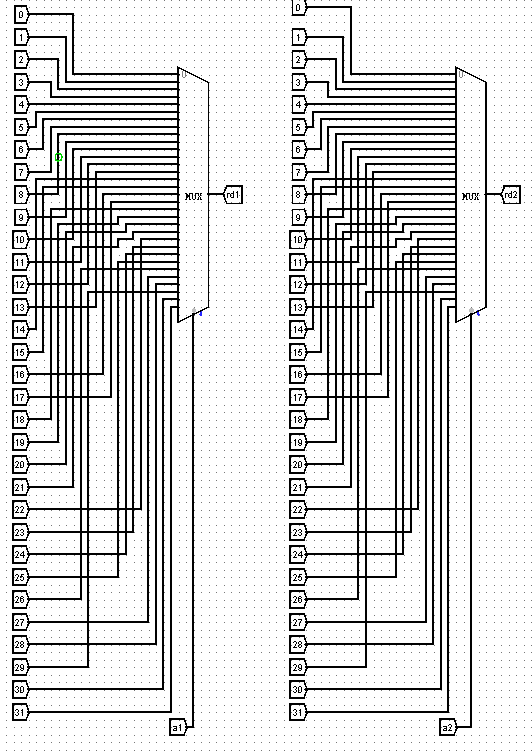
模块接口

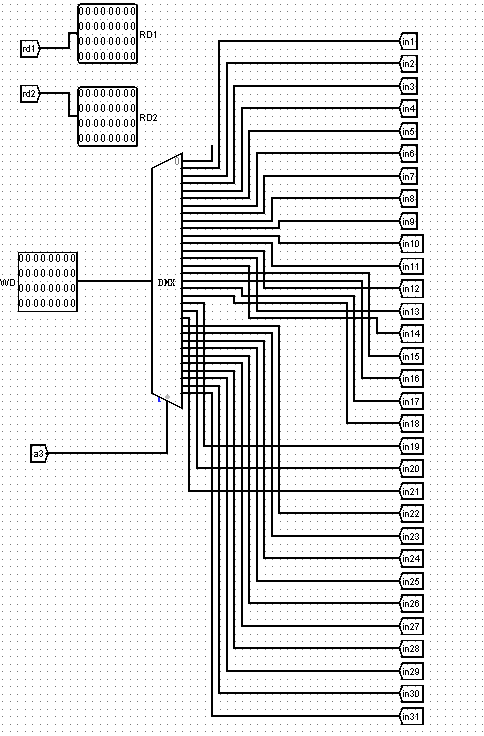
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| WD[31:0] | I | 写入数据的输入 |
| RA1[4:0] | I | 读寄存器地址 1 |
| RA2[4:0] | I | 读寄存器地址 2 |
| WA[4:0] | I | 写寄存器地址 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号  1：复位  0：无效 |
| RegWrite | I | 是否可以写入控制信号(随时都可以读出)  1：可以写  0：不可以写 |
| RD1[31:0] | O | 32 位数据输出 1 |
| RD2[31:0] | O | 32 位数据输出 2 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，所有寄存器被设置为 0x00000000 |
| 2 | 读寄存器 | 根据输入的寄存器地址读出32位数据 |
| 3 | 写寄存器 | 根据输入的地址，把输入的数据写进所选的寄存器 |







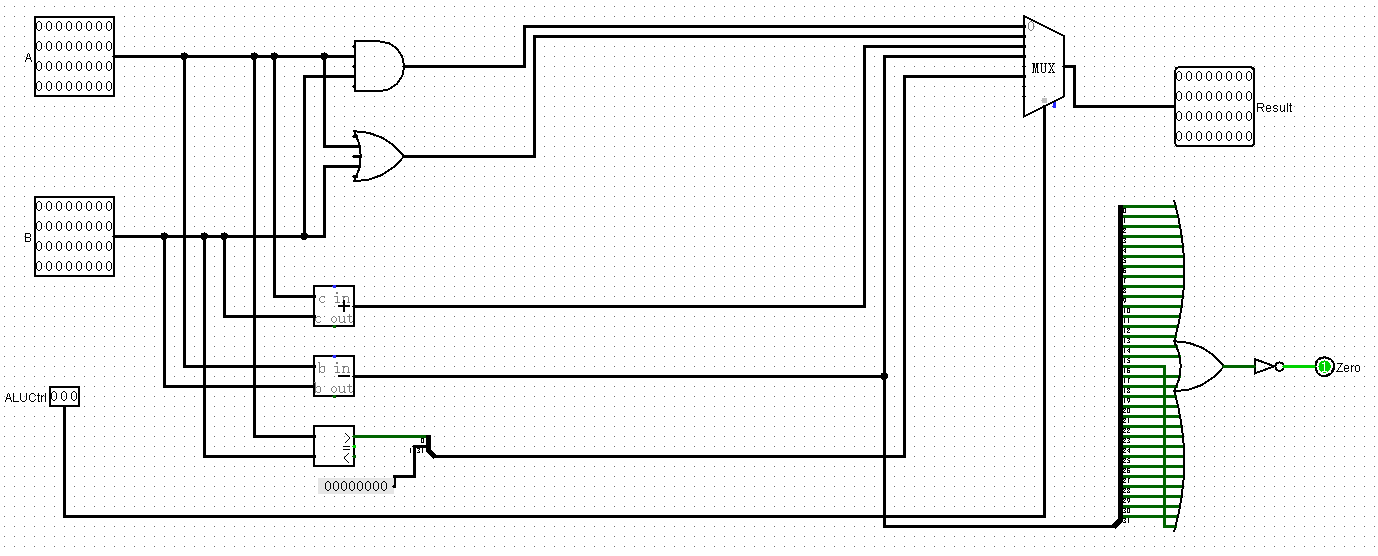
3.ALU

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| A[31:0] | I | 32 位输入数据 1 |
| B[31:0] | I | 32 位输入数据 2 |
| ALUCtrl[2:0] | I | 控制信号  000：与  001：或  010：加  011：减  101：比较大小 |
| Result[31:0] | O | 32 位数据输出 |
| Zero | O | A,B是否相等的标志信号  1：相等  0：不相等 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 与 | A&B |
| 2 | 或 | A|B |
| 3 | 加 | A+B |
| 4 | 减 | A-B |
| 5 | 判零 | A=?B |
| 6 | 比较大小 | A>B?A<B |



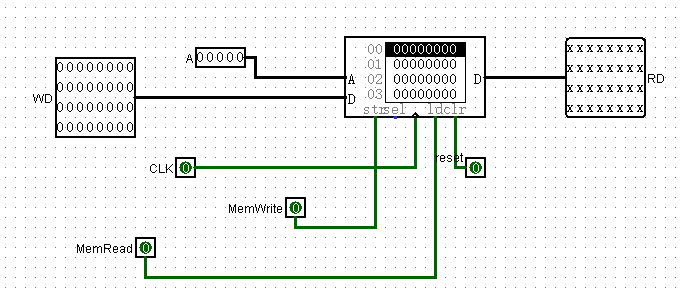
4.DM

模块接口

|  |  |  |
| --- | --- | --- |
| 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 |
| MemWrite | I | 读写控制信号  1：写操作 |
| MemRead | I | 读写控制信号  1：读操作 |
| A[4:0] | I | 操作寄存器地址 |
| WD[31:0] | I | 输入（写入内存）的32位数据 |
| RD[31:0] | O | 32 位数据输出 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 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 |



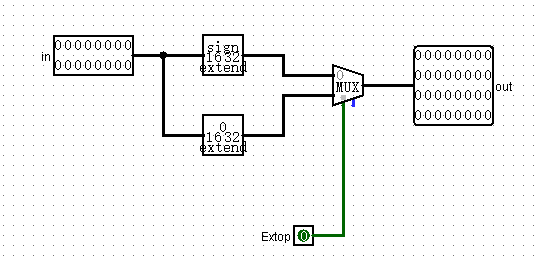
5. EXT

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| In[15:0] | I | 16 位数据输入 |
| Out[32:0] | O | 32 位数据输出 |
| ExtOp | I | 控制信号  0：高位符号扩展  1：高位0扩展 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 高位符号扩展 | 高16位补符号位 |
| 2 | 高位0扩展 | 高16位补0 |



6. Controller

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| Op[5:0] | I | 6位opcode段 |
| RegDst | O | 写地址控制 选择RT,RD |
| ALUSrc | O | ALU第二操作数选择控制 |
| RegWrite | O | GRF 写入控制 |
| MemRead | O | DM读信号 |
| MemWrite | O | DM写信号 |
| MemToReg[1:0] | O | GRF写入数据的选择信号 |
| ExtOp | O | 高位扩展方式选择信号 |
| Branch | O | 判断是否为beq指令的信号 是则为1 |
| ALUOp[2:0] | O | 传递给ALUControl与func[5:0]共同确定ALUctrl[3:0]的信号 |

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 |

7.ALUControl

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| ALUOp[2:0] | I | 由op[5:0]产生的控制信号 |
| Func[5:0] | I | 6位funcode |
| ALUCtrl[3:0] | O | ALU选择信号 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生ALU控制信号 | 产生ALU控制信号，控制ALU进行不同的运算 |

三．控制器设计思路

数据通路如下

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Adder | | PC | IM.A | GRF | | | | ALU | | DM | | EXT | Nadd | | Shift |
| A | B | RA1 | RA2 | WA | WD | ALU | B | A | WD | A | B |  |
| R型 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 |  |  |  |  |  |  |
| lw | PC | 4 | Adder | PC | Rs |  | Rt | DM.RD | RF.RD1 | sign\_ext | ALU |  | imm16 |  |  |  |
| sw | PC | 4 | Adder | PC | Rs | Rt |  |  | RF.RD1 | sign\_ext | ALU | RF.RD2 | imm16 |  |  |  |
| beq | PC | 4 | Adder  /Nadd | PC | Rs | Rt |  |  | RF.RD1 | RF.RD2 |  |  | imm16 | Adder | Shift | Sign\_ext |
| ori | PC | 4 | Adder | PC | Rs |  | Rt | ALU | RF.RD1 | zero\_ext |  |  | imm16 |  |  |  |
| lui | PC | 4 | Adder | PC |  |  | Rt | imm+016 |  |  |  |  |  |  |  |  |
| nop | PC | 4 | Adder | PC |  |  |  |  |  |  |  |  |  |  |  |  |

由此可见需要以下几个MUX多路选择器

1.beq指令 PC有两种选择 PC=Adder输出或者Nadd的输出 选择信号为Branch

2.GRF的WA端选择Rd,Rt需要一个MUX，控制信号RegDst

3.GRF的WD输入端，有三种选择：RF.RD2，ALU的输出，lui指令直接对imm16后边补16位0，需要2选4MUX,选择信号MemToReg[1:0]

4.两种扩展方式的选择（符号扩展，0扩展）选择信号EXTOp

5.ALU的B端两种选择，RF.RD2或EXT的输出，选择信号ALUSrc

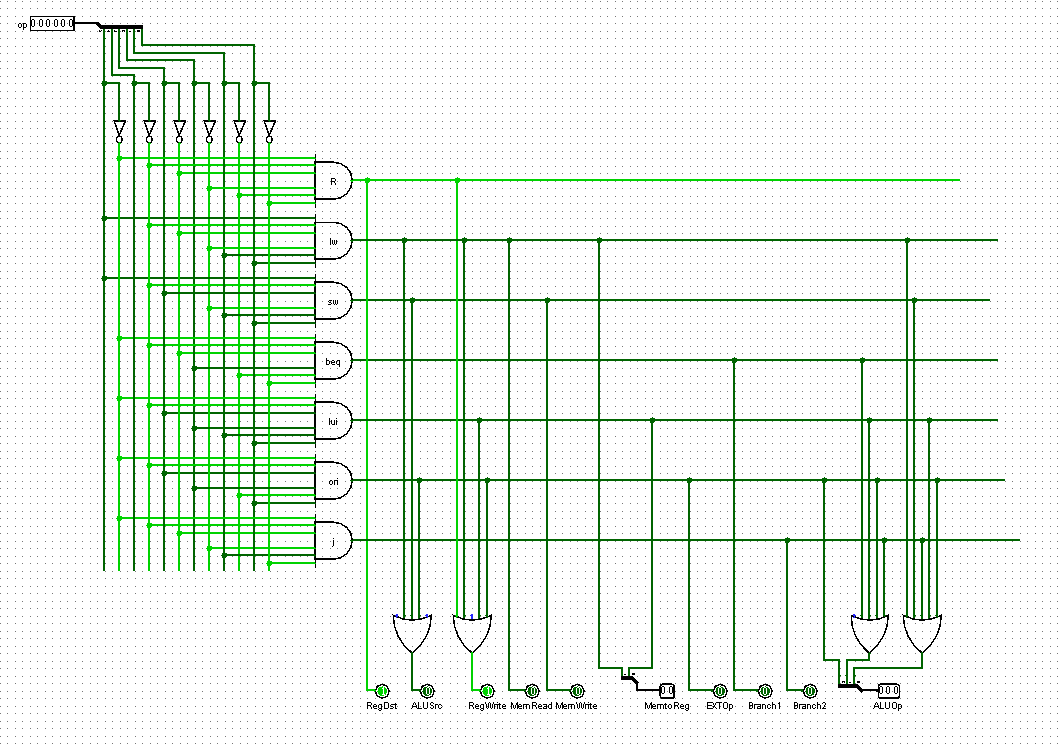
除了上述Branch, ALUSrc, EXTOp, MemToReg[1:0], RegDst, 还有三个读写控制信号，RegWrite是GRF写入信号，MemRead, MemWrite是DM读写信号，所以控制器Controller需要设计这8个控制信号。除此之外，设定ALUOp[2:0]用以作为ALUController的输入，与func[5:0]一起，决定ALU的控制信号ALUCtrl[2:0],即决定ALU的运算方式（加减与或）。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 功能描述 |
| Op[5:0] | I | 6位opcode段 |
| RegDst | O | 写地址控制 选择RT,RD |
| ALUSrc | O | ALU第二操作数选择控制 |
| RegWrite | O | GRF 写入控制 |
| MemRead | O | DM读信号 |
| MemWrite | O | DM写信号 |
| MemToReg[1:0] | O | GRF写入数据的选择信号 |
| ExtOp | O | 高位扩展方式选择信号 |
| Branch | O | 判断是否为beq指令的信号 是则为1 |
| ALUOp[2:0] | O | 传递给ALUControl与func[5:0]共同确定ALUctrl[3:0]的信号 |

可以绘制如下表格

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| name | R | lw | sw | beq | lui | ori | nop |
| Op5 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| Op4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Op3 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| Op2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| Op1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| Op0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| RegDst | 1 | 0 | x | x | 0 | 0 | x |
| ALUSrc | 0 | 1 | 1 | 0 | x | 1 | x |
| RegWrite | 1 | 1 | 0 | 0 | 1 | 1 | x |
| MemRead | 0 | 1 | 0 | 0 | 0 | 0 | x |
| MemWrite | 0 | 0 | 1 | 0 | 0 | 0 | x |
| MemToReg[1:0] | 00 | 10 | x0 | x0 | x1 | 00 | xx |
| EXTOp | 0 | 0 | 0 | 0 | 0 | 1 | x |
| Branch | 0 | 0 | 0 | 1 | 0 | 0 | x |
| ALUOp[2:0] | 000 | 001 | 001 | 010 | 011 | 111 | xxx |

与或门阵列为

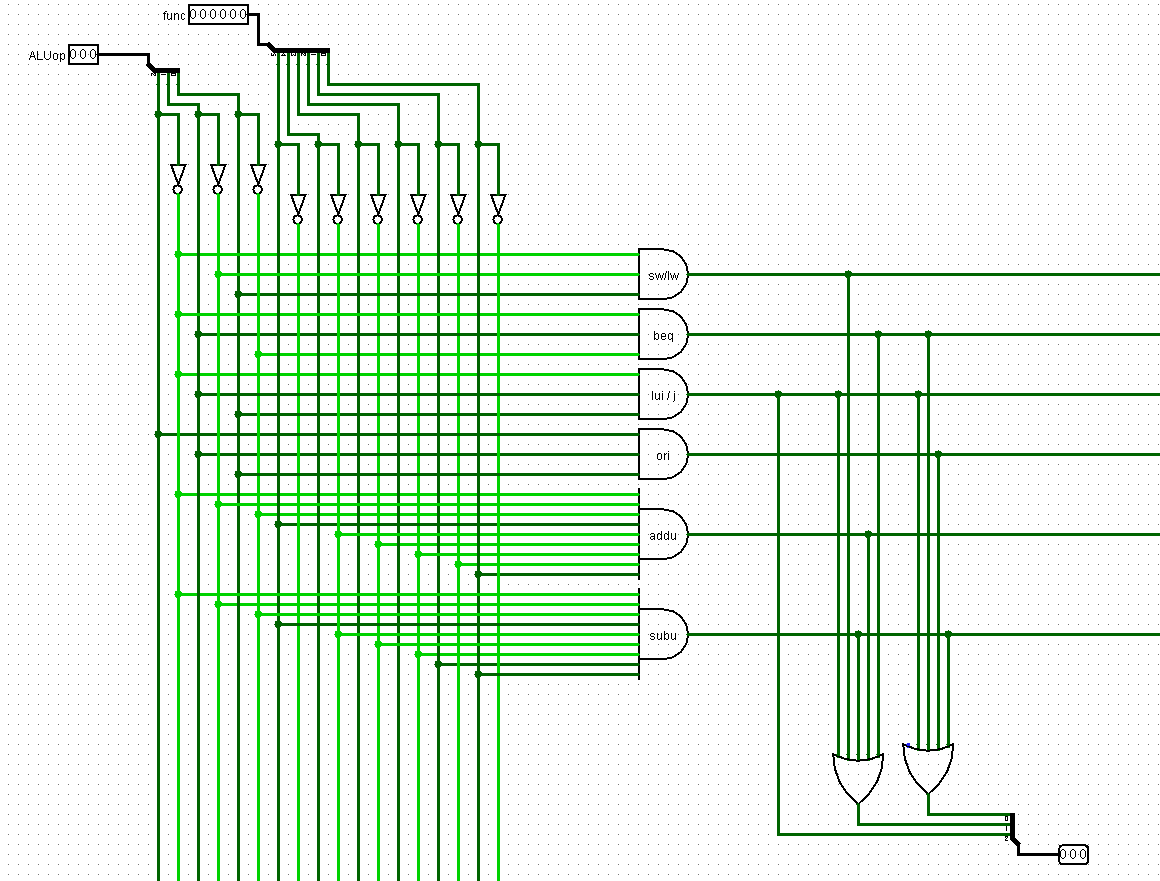


ALUController的设计

R类型的ALUCtrl需要关心Func[5:0],而其他类型只关心由Controller产生的ALUOp[2:0],根据实际意义确定ALU的运算类型，画表入下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | Func[5:0] | ALUOp[2:0] | 类型 | ALUCtrl[2:0] |
| lw | xxxxxx | 001 | 加 | 010 |
| sw | xxxxxx | 001 | 加 | 010 |
| beq | xxxxxx | 010 | 减 | 011 |
| lui | xxxxxx | 011 | x | 111 |
| ori | xxxxxx | 111 | 或 | 001 |
| addu | 100001 | 000 | 加 | 010 |
| subu | 100011 | 000 | 减 | 011 |

与或门阵列



四．测试程序

ori $a0,$0,1999 #ori 测试程序要实现: $0寄存器中的内容与立即数 0x000007cf进行或运算，储存在$a0寄存器中

ori $a1,$a0,111 #ori 测试程序要实现: $a0寄存器中的内容与立即数 0x0000006f进行或运算，储存在$a1寄存器中

lui $a2,12345 #lui 测试程序要实现: 立即数 0x00003039 加载至 $a2 寄存器的高位

lui $a3,0xffff #lui 测试程序要实现: 立即数 0x0000ffff 加载至 $a3 寄存器的高位

ori $a3,$a3,0xffff #ori 测试程序要实现: $a3寄存器中的内容与立即数 0x0000ffff进行或运算，储存在$a3寄存器中

addu $s0,$a0,$a1 #addu 测试程序要实现：a0 寄存器中的值加上a1 后存到 s0 寄存器中

addu $s1,$a3,$a3 #addu 测试程序要实现：a3 寄存器中的值加上a3 后存到 s1 寄存器中

addu $s2,$a3,$s0 #addu 测试程序要实现：a3 寄存器中的值加上s0 后存到 s2 寄存器中

subu $s0,$a0,$s2 #subu 测试程序要实现：a0 寄存器中的值减去 s2 寄存器中的值后存到 s0 寄存器中

subu $s1,$a3,$a3 #subu 测试程序要实现：a3 寄存器中的值减去 a3 寄存器中的值后存到 s1 寄存器中

eee:

subu $s2,$a3,$a0 #subu 测试程序要实现：a3 寄存器中的值减去 a0 寄存器中的值后存到 s2 寄存器中

subu $s3,$s2,$s1 #subu 测试程序要实现：s2 寄存器中的值减去 s1 寄存器中的值后存到 s3 寄存器中

ori $t0,$0,0x0000 #ori 测试程序要实现: $0寄存器中的内容与立即数 0x00000000进行或运算，储存在$t0寄存器中

sw $a0,0($t0) #sw 测试程序要实现：把 a0 寄存器中值,存储到t0寄存器的值再加上偏移量 0， 所指向的 RAM 中

nop

sw $a1,4($t0) #sw 测试程序要实现：把 a1 寄存器中值,存储到t0寄存器的值再加上偏移量 4， 所指向的 RAM 中

sw $s0,8($t0) #sw 测试程序要实现：把 s0 寄存器中值,存储到t0寄存器的值再加上偏移量 8， 所指向的 RAM 中

sw $s1,12($t0) #sw 测试程序要实现：把 s1 寄存器中值,存储到t0寄存器的值再加上偏移量 12， 所指向的 RAM 中

sw $s2,16($t0) #sw 测试程序要实现：把 s2 寄存器中值,存储到t0寄存器的值再加上偏移量 16， 所指向的 RAM 中

lw $t7,0($t0) #lw 测试程序要实现：把 t0 寄存器的值加上偏移量0 当作地址读取存储器中的值存入 t7

lw $t6,20($t0) #lw 测试程序要实现：把 t0 寄存器的值加上偏移量20 当作地址读取存储器中的值存入 t6

sw $t6,24($t0) #sw 测试程序要实现：把 t6 寄存器中值,存储到t0寄存器的值再加上偏移量 24， 所指向的 RAM 中

lw $t5,12($t0) #lw 测试程序要实现：把 t0 寄存器的值加上偏移量12 当作地址读取存储器中的值存入 t5

ori $t0,$t0,1 #ori 测试程序要实现: $t0寄存器中的内容与立即数 0x00000001进行或运算，储存在$t0寄存器中

ori $t1,$t1,1 #ori 测试程序要实现: $t1寄存器中的内容与立即数 0x00000001进行或运算，储存在$t1寄存器中

ori $t2,$t2,2 #ori 测试程序要实现: $t2寄存器中的内容与立即数 0x00000002进行或运算，储存在$t2寄存器中

beq $t0,$t2,eee #beq 测试程序要实现：判断 t0 的值和 t2 的值是否相等，相等转eee

beq $t0,$t1,end #beq 测试程序要实现：判断 t0 的值和 t1 的值是否相等，相等转end

lui $t3,1111 #lui 测试程序要实现: 立即数 0x00000457 加载至 $t3 寄存器的高位

end:

addu $t0,$t0,$t7 #addu 测试程序要实现：t0 寄存器中的值加上t0 后存到 t0 寄存器中

机器码

v2.0 raw

340407cf

3485006f

3c063039

3c07ffff

34e7ffff

00858021

00e78821

00f09021

00928023

00e78823

00e49023

02519823

34080000

ad040000

00000000

ad050004

ad100008

ad11000c

ad120010

8d0f0000

8d0e0014

ad0e0018

8d0d000c

35080001

35290001

354a0002

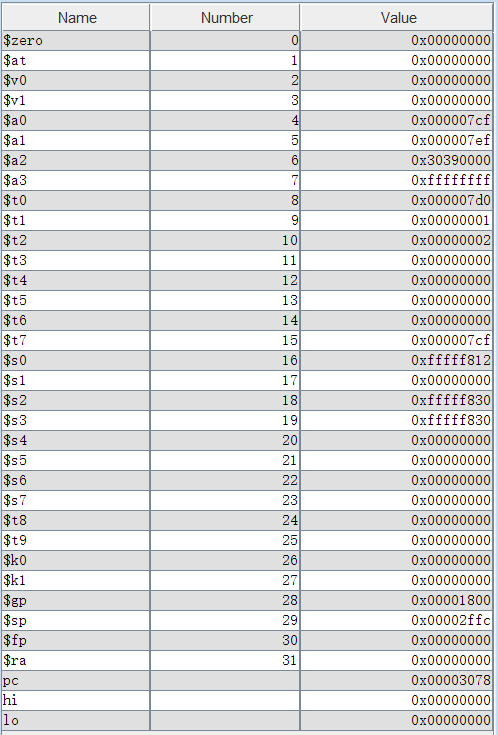
110affef

11090001

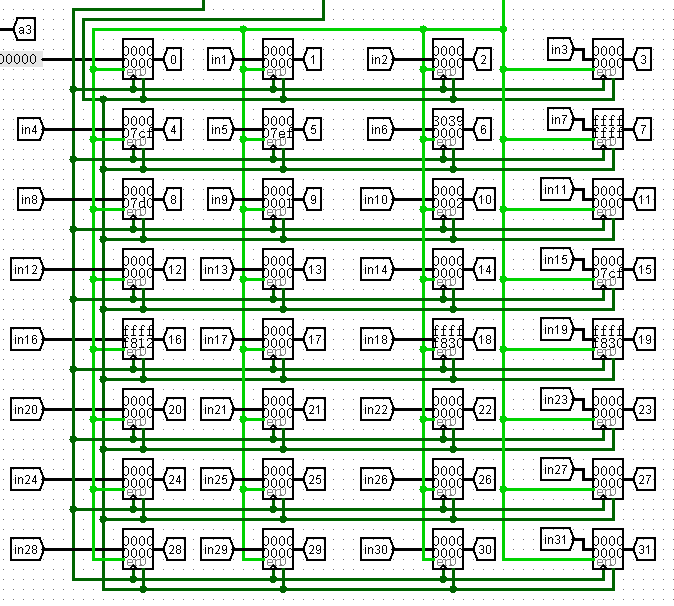
3c0b0457

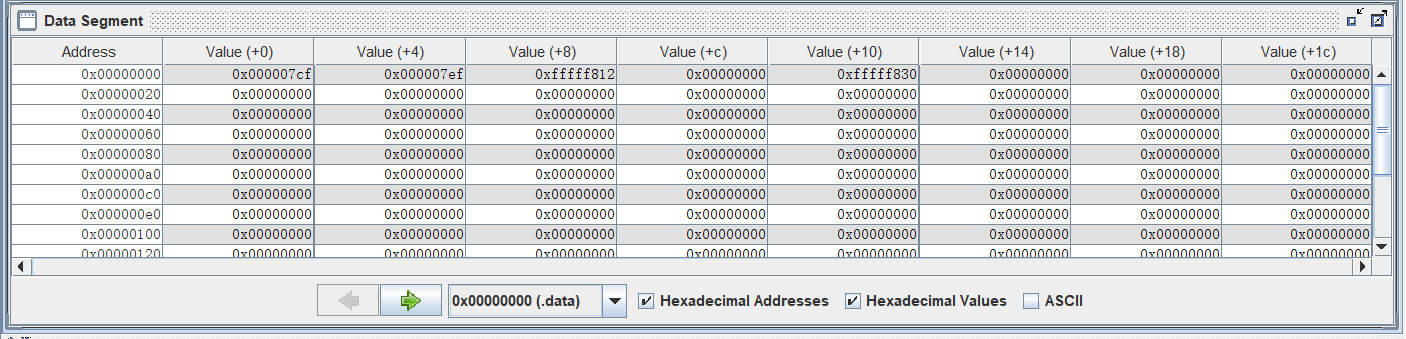
010f4021

MARS模拟结果如下

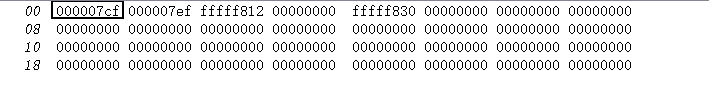


GRF





DM



五．思考题

L0.T2

1.若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

32位PC可以直接进行PC+4，不用扩展,30位PC需要扩展

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。

合理，ROM是只读存储器，作为IM指令存储器来存储指令，只负责读出指令，而DM用来想内存中存储数据，可读可写，不需要速度很快，但需要足够的存储空间，RAM正适合，而GRF是寄存器堆，速度要快，选择寄存器，寄存器是这三种原件中速度最快的了，所以很合适。

L0.T3

1.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。）

Func f5,f4,f3,f2,f1

Op o5,o4,o3,o2,o1

与&& 或|| 非！

RegDst=!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0

ALUSrc=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)||(o5&&!o4&&o3&&!o2&&o1&&o0)

MemtoReg=o5&&!o4&&!o3&&!o2&&o1&&o0

RegWrite=(!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0)||(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)

nPC\_Sel=!o5&&!o4&&!o3&&o2&&!o1&&!o0

ExtOp=o5&&!o4&&!o2&&o1&&o0

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。

将 X 按照方便化简的原则当成 0 或 1

RegDst=!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0

ALUSrc=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)||(o5&&!o4&&o3&&!o2&&o1&&o0)=(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o2&&o1&&o0)

MemtoReg=o5&&!o4&&!o3&&!o2&&o1&&o0

RegWrite=(!o5&&!o4&&!o3&&!o2&&!o1&&f5&&!f4&&!f3&&!f2&&!f0)||(!o5&&!o4&&o3&&o2&&!o1&&o0)||(o5&&!o4&&!o3&&!o2&&o1&&o0)

nPC\_Sel=!o5&&!o4&&!o3&&o2&&!o1&&!o0

ExtOp=o5&&!o4&&!o2&&o1&&o0

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是0X00000000,只是做了PC=PC+4，别的什么都没有修改，没对逻辑电路电路中的元件进行任何操作，存在与否对电路没有影响。

L0.T4

1.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。

假设DM有256MB容量,并且映射在0x3000\_0000-0x3FFF\_FFFF区间,那么只需要把高4位地址与0x3进行比较,结果就是DM的片选信号。之前的 DM存满后就从 0x3000\_0000~0x3FFF\_FFFF存储数据。这次的 DM 最多存到0x0000\_00fc,所以不需要片选信号。

2.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。

所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。

1.组合逻辑电路的逻辑验证

(1)转换为单一抽象模型比较。通过对单一表示的结构进行比较，得出其功能等价的结论。在最坏的情况下，布尔函数为正，表示随输入个数指数增加，其过大的内存需求限制了一般[布尔函数](https://baike.baidu.com/item/%E5%B8%83%E5%B0%94%E5%87%BD%E6%95%B0/2250418)的验证能力。

(2)利用测试输入向量进行验证。探寻使两个电路具有不同输出的输入测试向量，若存在这样的测试向量，则电路在功能上等价。在最坏情况下，这种方法需要穷举所有可能的输入测试矢量，运行时间又成为一个主要问题。

2．时序逻辑电路的验证

对一个[时序电路](https://baike.baidu.com/item/%E6%97%B6%E5%BA%8F%E7%94%B5%E8%B7%AF/4836638)而言，可以把它看成一个[有限状态机](https://baike.baidu.com/item/%E6%9C%89%E9%99%90%E7%8A%B6%E6%80%81%E6%9C%BA/2081914)(FSM，finite-state machine)。电路功能的等价可以用有限状态机的等价来判断。假定有两个状态机A和B，要对它们进行比较。直观的说，当A和B有相同的接口，而且从相同的初始状态出发，两者对有效输入值序列产生相同的输出值序列，则可以说A和B等价。

形式验证的优点：

1. 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，不需要开发测试激励。

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，因此有效地克服了测试验证的不足。对指定描述的所有可能的情况进行验证，覆盖率达到了100%。

3. 形式验证可以进行从系统级到门级的验证，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，缩短设计周期。

形式验证的缺点：

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