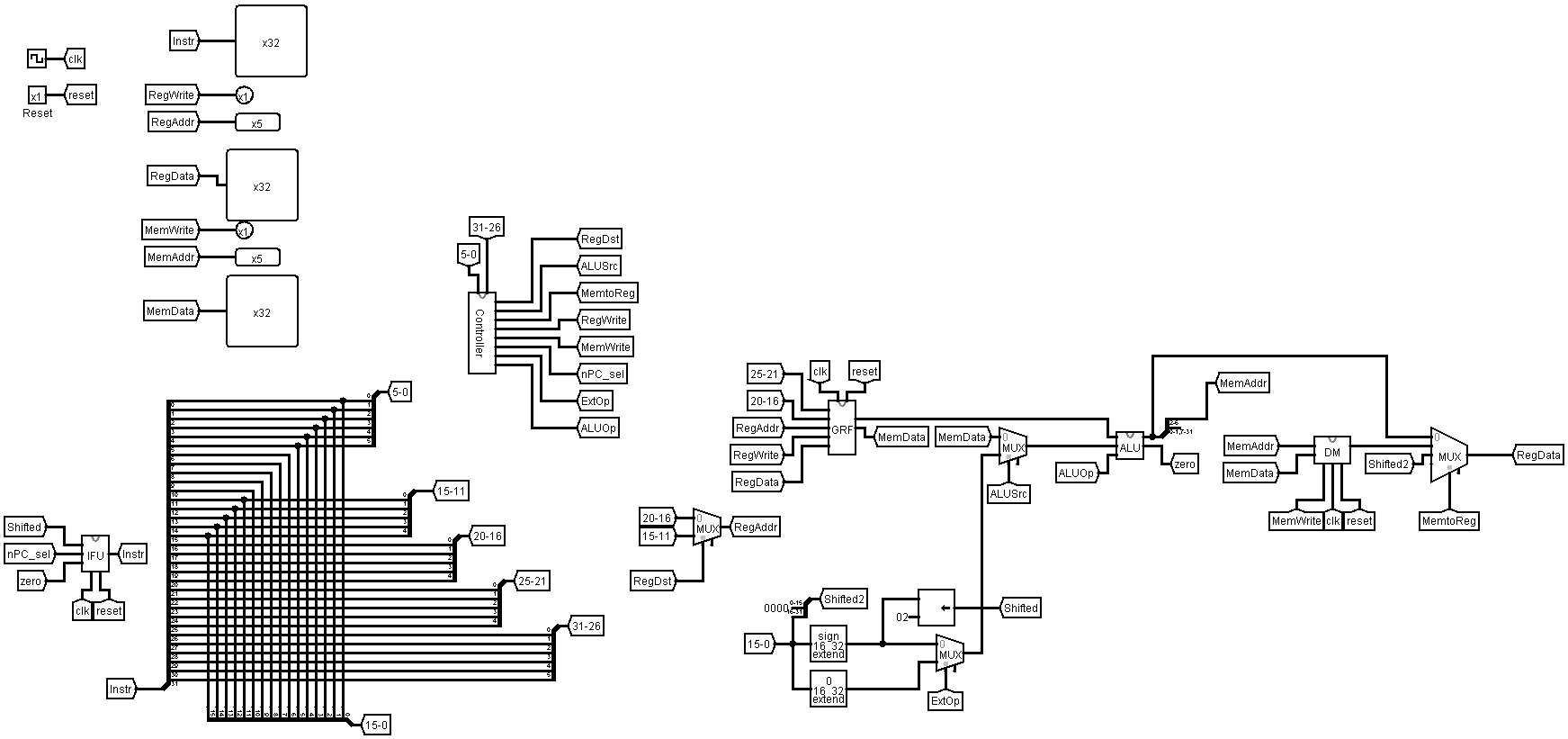
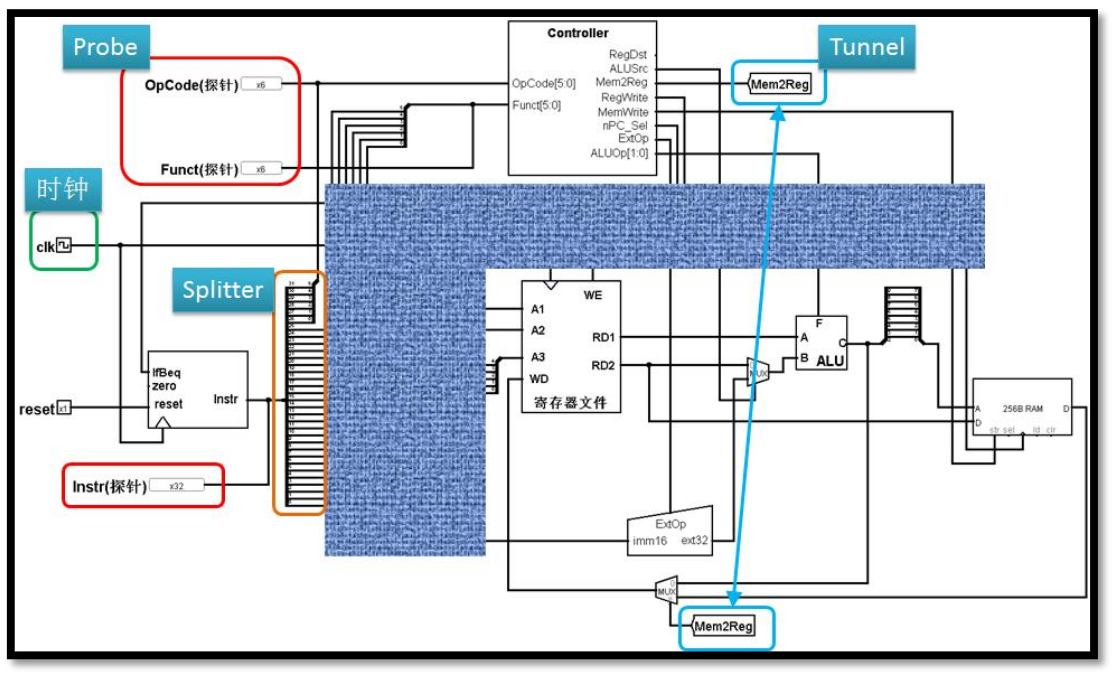
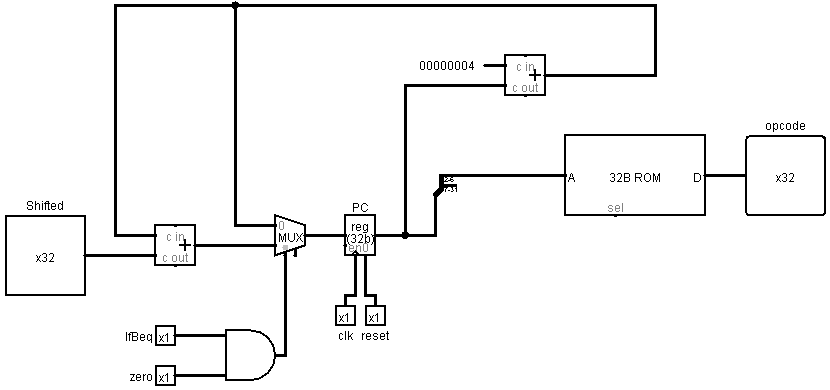
计算机组成课程设计

P3 课下测试 – Logisim单周期

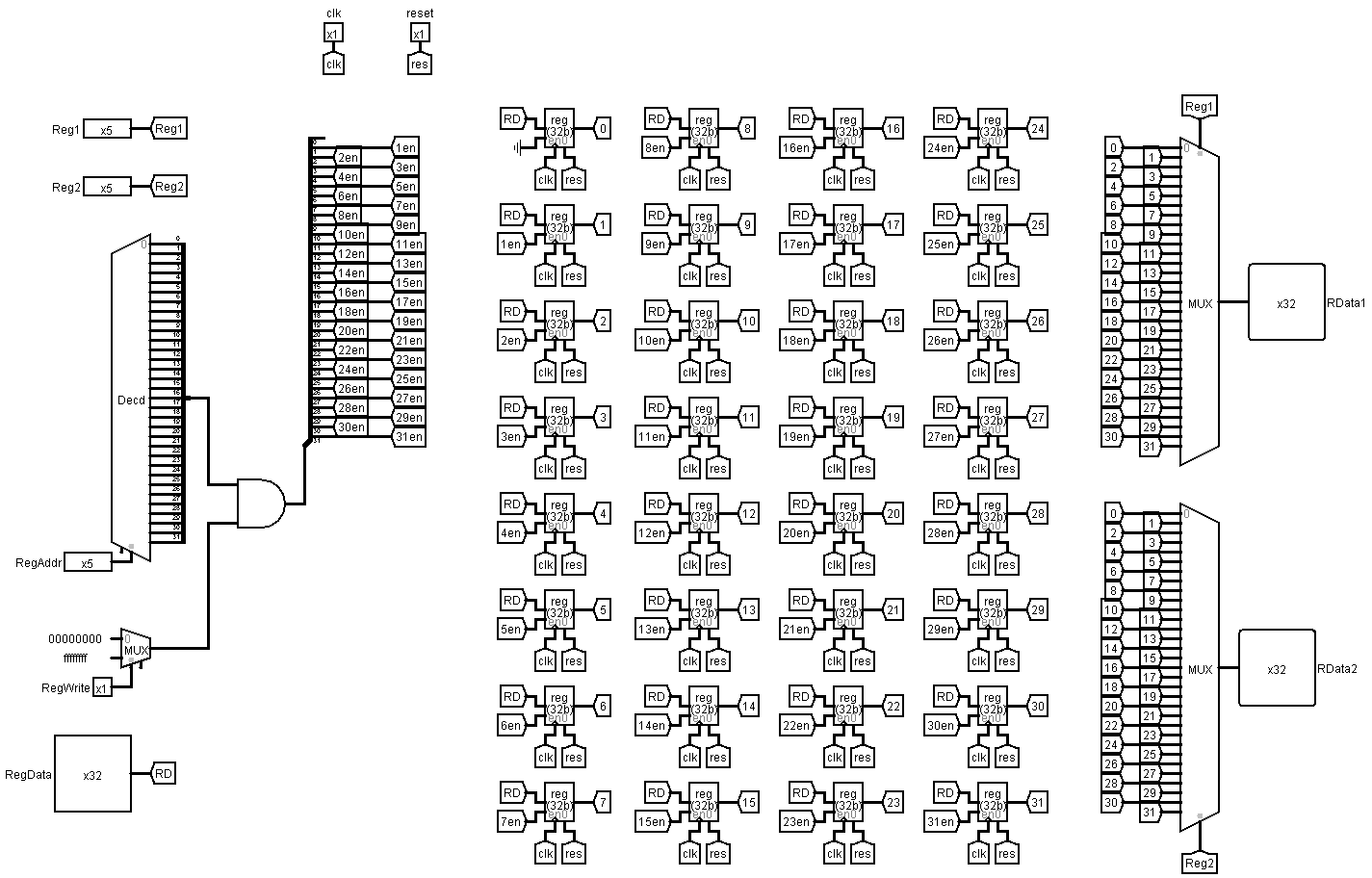
周美廷-76066002

1. 顶层设计
2. 模块规格
3. IFU（取指令单元）：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，应具有复位功能。
* 起始地址：0x00000000。
* IM用 ROM 实现，容量为 32bit \* 32。
* 因 IM 实际地址宽度仅为 5 位，故需要使用恰当的方法将 PC 中储存的地址同 IM 联系起来。

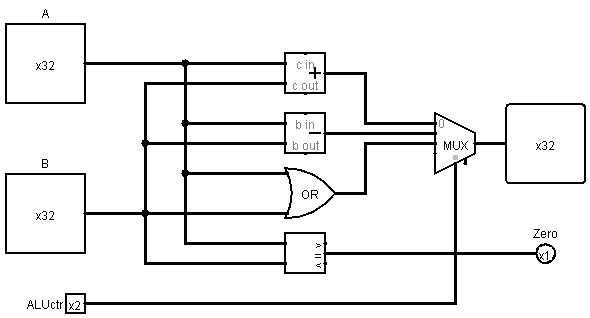
|  |  |  |
| --- | --- | --- |
| 功能名称 | 方向 | 功能描述 |
| IfBeq | I | 当前置零是否为beq  1：指令为beq  0：指令不为beq |
| Shifted | I | 第0-15指令被Extend后移植 |
| zero | I | ALU计算是否为0  1：为0  0：不为0 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| opcode[31:0] | O | 32位MIPS指令 |

1. GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

* 用具有写使能的寄存器实现，寄存器总数为 32 个。
* 0 号寄存器的值始终保持为 0。其他寄存器初始值均为 0，无需专门设置。

|  |  |  |
| --- | --- | --- |
| 功能名称 | 方向 | 功能描述 |
| RegData[31:0] | I | 写入数据 |
| RegWrite | I | 读写控制信号  1：写  0：读 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| RegAddr[4:0] | I | 写寄存器地址 |
| Reg1[4:0] | I | 读寄存器地址1 |
| Reg2[4:0] | I | 读寄存器地址2 |
| RData1[31:0] | O | 32位数据1 |
| RData2[31:0] | O | 32位数据2 |

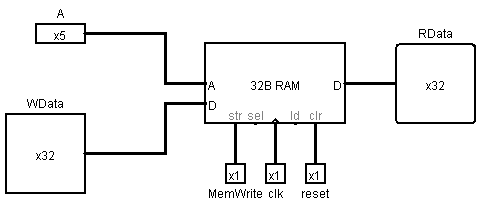
1. ALU（算术逻辑单元）

* 提供 32 位加、减、或运算及大小比较功能。
* 可以不支持溢出（不检测溢出）。

|  |  |  |
| --- | --- | --- |
| 功能名称 | 方向 | 功能描述 |
| A[31:0] | I | 输入数据A |
| B[31:0] | I | 输入数据B |
| ALUctr[1:0] | I | ALU控制信号  00：加法运算  01：减法运算  10：或运算  11：比较 |
| zero | O | 计算结果是否为0 |
| Output[31:0] | O | 输出结果32位 |

1. DM（数据存储器）

* 使用 RAM 实现，容量为 32bit \* 32。
* 起始地址：0x00000000。
* RAM 应使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。

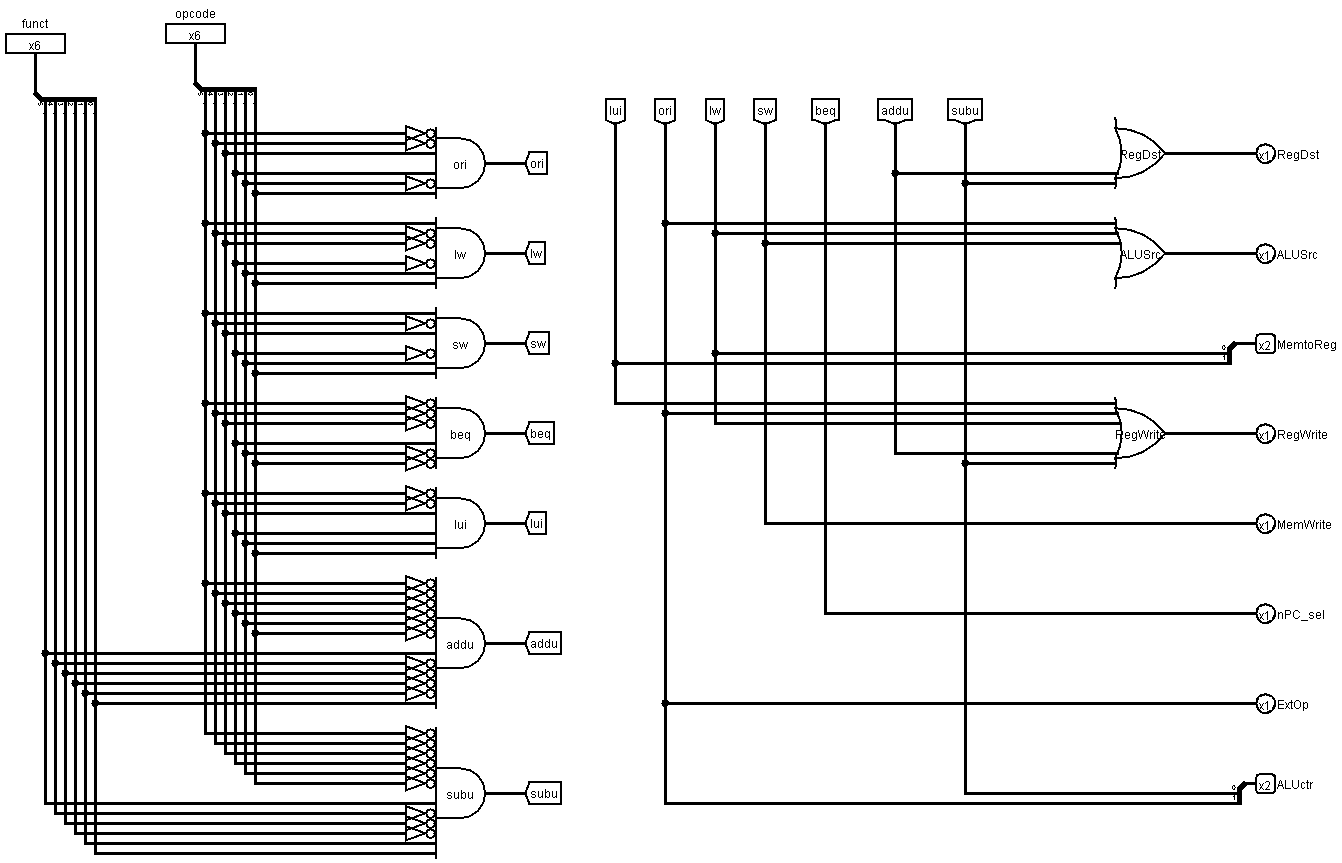


|  |  |  |
| --- | --- | --- |
| 功能名称 | 方向 | 功能描述 |
| A[4:0] | I | MemAddr地址 |
| WData[31:0] | I | 输入数据32位 |
| MemWrite | I | 读写控制信号 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| RData[31:0] | O | 输出结果32位 |

1. EXT

* 可以使用 logisim 内置的 Bit Extender。

1. Controller（控制器）

* 使用与或门阵列构造控制信号
* 具体方法看后文

|  |  |  |
| --- | --- | --- |
| 功能名称 | 方向 | 功能描述 |
| funct[5:0] | I | Function 6位 |
| opcode[5:0] | I | Opcode 6位 |
| RegDst | O | 写地址控制 |
| ALUSrc | O | 控制ALU |
| MemtoReg[1:0] | O | DM读控制 |
| RegWrite | O | GRF读写控制 |
| MemWrite | O | DM写控制 |
| nPC\_sel | O | beq指令标志 |
| ExtOp | O | Ext扩展控制 |
| ALUctr[1:0] | O | ALU运算操作控制 |

1. 控制器设计

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| funct | 100001 | 100011 | n/a | | | | | 000000 |
| opcode | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000000 |
|  | addu | subu | ori | lw | sw | beq | lui | nop |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 | x |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | x | x |
| MemtoReg[1:0] | 00 | 00 | 00 | 01 | xx | xx | 10 | xx |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| nPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ExtOp | x | x | 1 | 0 | 0 | x | 0 | x |
| ALUctr[1:0] | Add | Subtract | Or | Add | Add | xx | xx | xx |

1. 测试程序说明

测试指令集：

.data

testarr: .space 8

.text

lui $t0, 0x1000 #Load upper immediate (of high-order 16-bit),write 0x10000000 to $t0

ori $t1, $t0, 0x0001 #Set $t1 to bitwise OR of $t0 and 0x1000001

nop #Null op

addu $t2, $t1, 3 #Set $t2 to 0x10000004

subu $t3, $t1, $t2 #Set $t3 to (ALU operation $t1 - $t2), no overflow (nothing changed)

nop #Null op

sw $t3, testarr($zero) #Write $t3’s value to DM[testarr+$zero] 0x10000000

beq $t3, $t2, any #ALU checks if GRF[t3]==GRF[t2], if yes then goto any, else PC+4

lw $t3, testarr($zero) #Load DM[testarr+$zero] to GRF[t2] 0x10000000

nop #Null op

any: ori $t0, $t0, 0x1010 #Set $t0 to bitwise OR of itself ($t0) and 0x1000001 -> 0x10001010

机器码：

3c081000

35090001

00000000

3c010000

34210003

01215021

3c010000

01215823

00000000

ac0b0000

116a0002

8c0b0000

00000000

35081010

1. 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。

**优势：**32位PC可以访问PC+4，比30位的访问的空间多。

**劣势：**32位PC的设计比30位PC的更复杂。由于32位PC的低两位都是0，如果用30位PC的话，每次PC+1即可，相当于32位PC的高30位

1. 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。

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，有可能会有更新，所以用RAM

GRF是临时存储单元，所以用寄存器。

(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。）

𝑅𝑒𝑔𝐷𝑠𝑡 = 𝐹𝑢𝑛𝑐5 𝐹𝑢𝑛𝑐0

𝑅𝑒𝑔𝑊𝑟𝑖𝑡𝑒 = 𝐹𝑢𝑛𝑐5 𝐹𝑢𝑛𝑐0 + 𝑂𝑝3 𝑂𝑝2 𝑂𝑝1 𝑂𝑝0

+ 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝐴𝐿𝑈𝑆𝑟𝑐 = 𝑂𝑝3 𝑂𝑝2 𝑂𝑝0 + 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝑀𝑒𝑚𝑡𝑜𝑅𝑒𝑔 = 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝑛𝑃𝐶\_\_𝑆𝑒𝑙 = 𝑂𝑝2

𝐸𝑥𝑡𝑂𝑝 = 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

1. 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。

𝑅𝑒𝑔𝐷𝑠𝑡 = 𝐹𝑢𝑛𝑐5 𝐹𝑢𝑛𝑐0

𝑅𝑒𝑔𝑊𝑟𝑖𝑡𝑒 = 𝐹𝑢𝑛𝑐5 𝐹𝑢𝑛𝑐0 + 𝑂𝑝3 𝑂𝑝2 𝑂𝑝1 𝑂𝑝0

+ 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝐴𝐿𝑈𝑆𝑟𝑐 = 𝑂𝑝3 𝑂𝑝2 𝑂𝑝0 + 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝑀𝑒𝑚𝑡𝑜𝑅𝑒𝑔 = 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

𝑛𝑃𝐶\_\_𝑆𝑒𝑙 = 𝑂𝑝2

𝐸𝑥𝑡𝑂𝑝 = 𝑂𝑝5 𝑂𝑝1 𝑂𝑝0

1. 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。

因为nop的机器码是0x00000000（全0）。所以nop执行的时候，MemWrite,WriteEn等要么是0，要么是x。

(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为256MB，在0x30000000~0x3FFFFFFF区间，在做选择时，只需要当前指令取出来。ALU运算后得到的地址的高4位与0x3作比较(无符号)，大于则片选信号为1，小于则为0。

1. 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。

**优势：**

1. 形式验证是用数学方法将待验证电路和功能描述或参考设计直接进行比较，所以测试者不必考虑如何获得测试向量
2. 形式验证对指定描述所有可能的情况进行验证，而不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. 形式验证可以短时间内进行从系统级到门级的验证，所以可以尽快发现和改正电路设计中的一些错误。

**劣势：**

形式验证还不能有效的验证电路的性能（电路的时延和功耗等）。