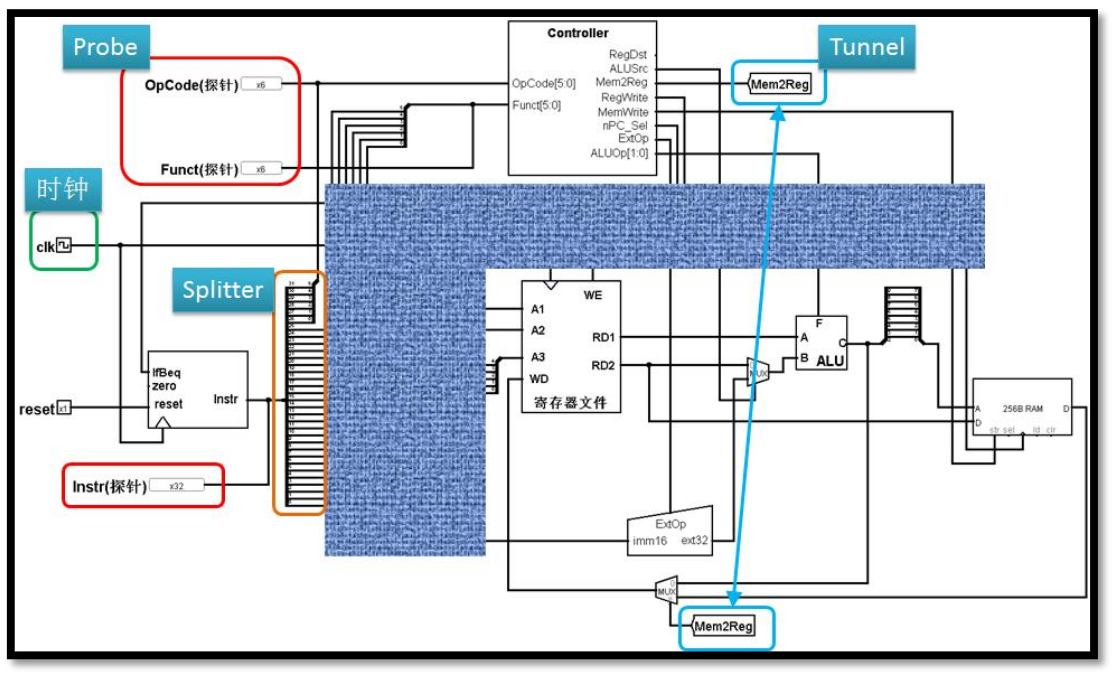
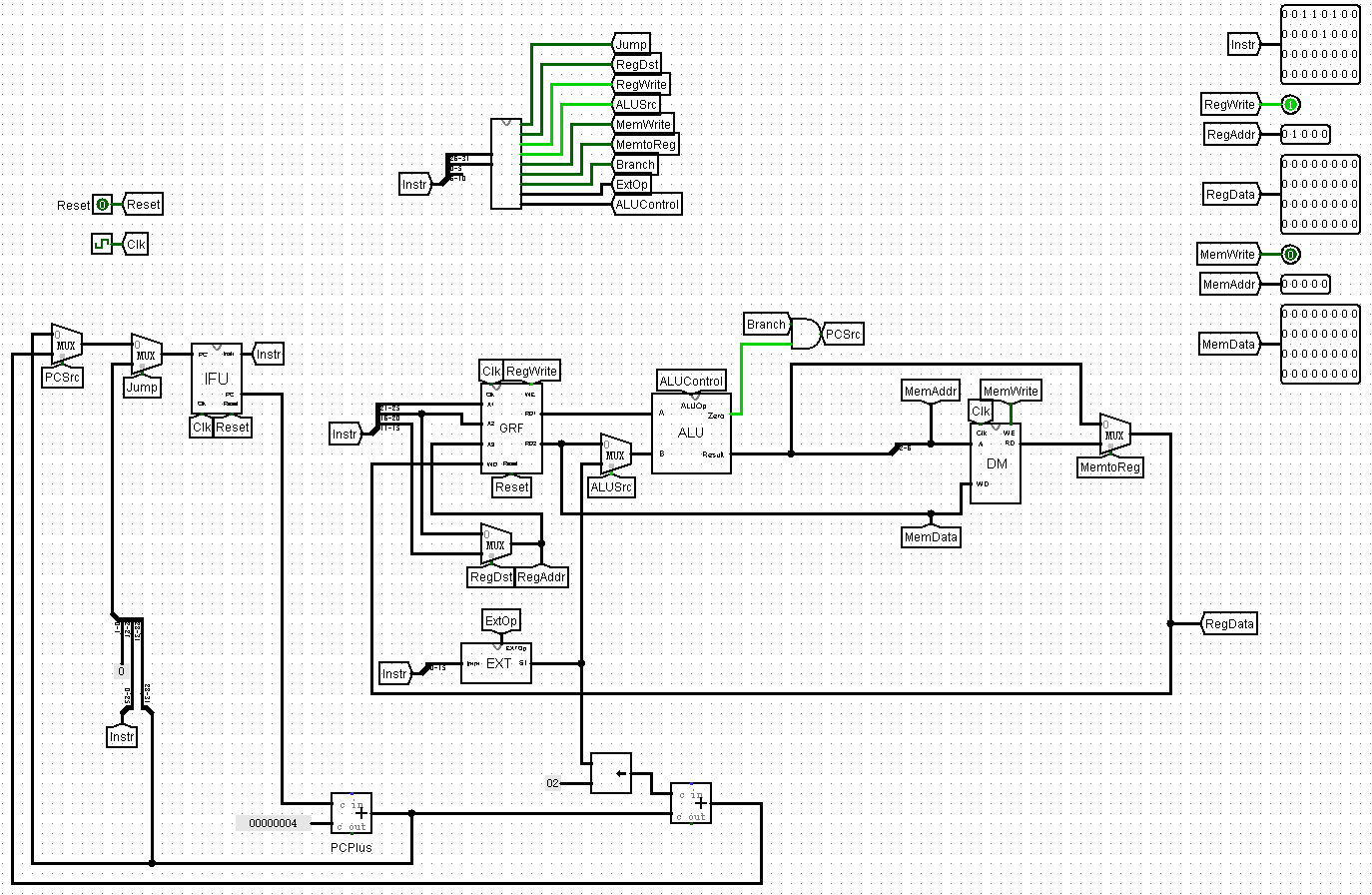
Project3 Logisim完成单周期处理器开发

1. 顶层设计视图





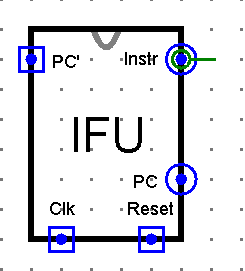
1. 模块定义
2. IFU（取指令单元）
3. 基本描述：
   1. PC 用寄存器实现，应具有复位功能。
   2. 起始地址：0x00000000。
   3. IM用 ROM 实现，容量为 32bit \* 32。
   4. 因 IM 实际地址宽度仅为 5 位，故需要使用恰当的方法将 PC 中储存的地址同 IM 联系起来。
4. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号  1：复位  0：无效 |
| PC’ | I | 更新PC值 |
| PC | O | 当前PC值 |
| Instr[31:0] | O | 指令存储器取出的32位指令 |

1. 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 |

1. 模块设计：



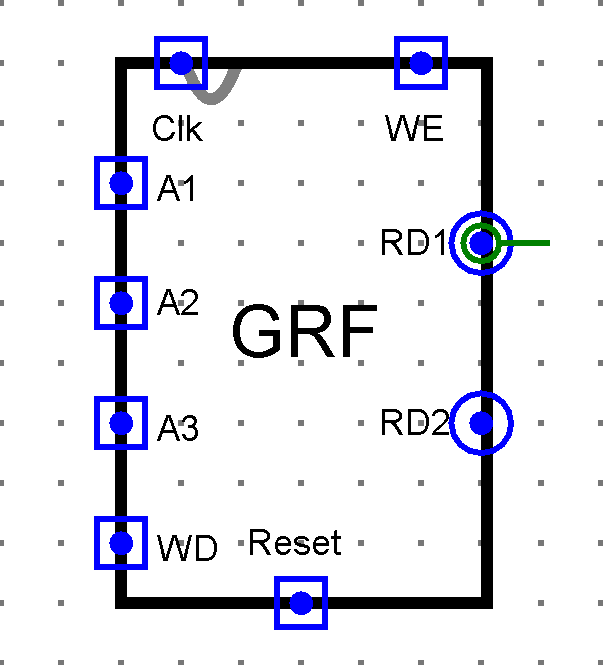
1. GRF（通用寄存器组，也称为寄存器文件、寄存器堆）
2. 基本描述：
   1. 用具有写使能的寄存器实现，寄存器总数为 32 。
   2. 0 号寄存器的值始终保持为 0。其他寄存器初始值均为 0，无需专门设置。
3. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 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 | 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[4:0] | I | 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[4:0] | I | 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 |
| A3[4:0] | I | 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[31:0] | I | 32位数据输入信号 |
| RD1[31:0] | O | 输出A1指定的寄存器中的32位数据 |
| RD1[31:0] | O | 输出A2指定的寄存器中的32位数据 |

1. 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RD1，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. 顶层设计：



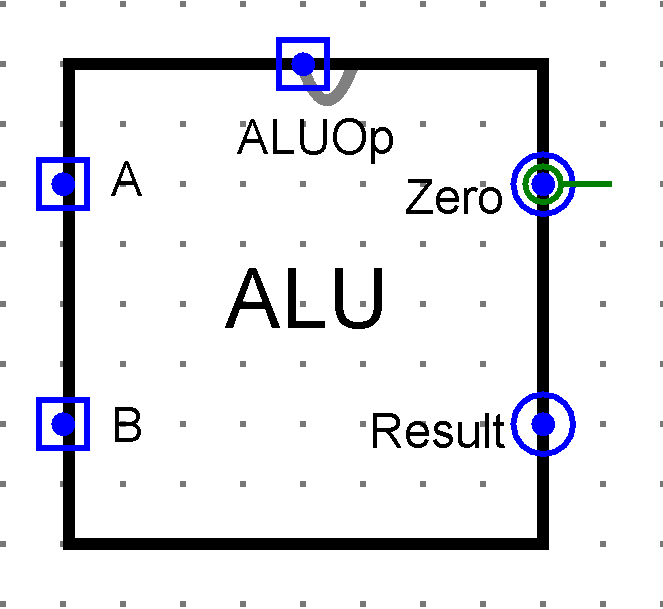
1. ALU（算术逻辑单元）
2. 基本描述：
   1. 提供 32 位加、减、或运算及大小比较功能。
   2. 可以不支持溢出（不检测溢出）。
3. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A [31:0] | I | 参与ALU计算的第一个值 |
| B [31:0] | I | 参与ALU计算的第二个值 |
| ALUOp [3:0] | I | ALU功能的选择信号  0000：加法运算  0001：减法运算  0010：或运算 |
| Zero | O | 零标志位 |
| Result [31:0] | O | ALU的32位运算结果 |

1. 功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | C = A + B |
| 2 | 减运算 | C = A - B |
| 3 | 或运算 | C = A | B |

1. 顶层设计：



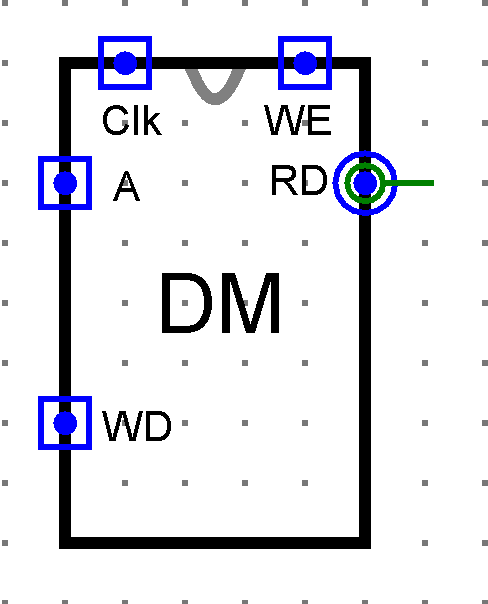
1. DM（数据存储器）
2. 基本描述：
   * 1. 使用 RAM 实现，容量为 32bit \* 32。
     2. 起始地址：0x00000000。
     3. RAM 应使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。
3. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| WE | I | 写使能信号 |
| A [4:0] | I | 5位地址输入信号 |
| WD [31:0] | I | 32位被写入存储器中的数据 |
| RD [31:0] | O | 32位被A中地址选择并输出的数据 |

1. 功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 读出地址A中所存储的数据到RD |
| 2 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入地址A |

1. 模块设计：



1. EXT
2. 基本描述：

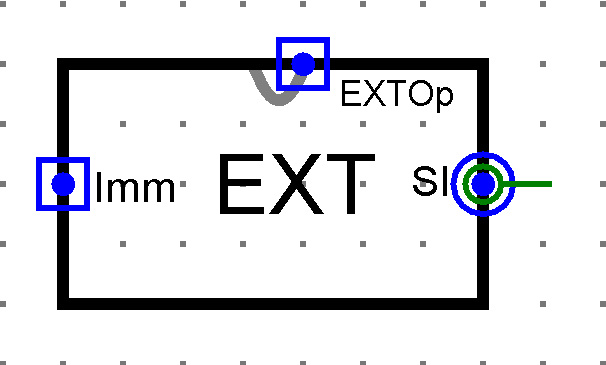
可以使用 logisim 内置的 Bit Extender。

1. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm [15:0] | I | 16位输入数据 |
| EXTOp [1:0] | I | 控制扩展方式  00：符号扩展  01：零扩展  10：加载至高位  11：符号扩展之后，左移两位 |
| SI [31:0] | O | 扩展后的32位输出数据 |

1. 功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 符号扩展 | 将16位输入数据符号扩展为32位输出数据 |

1. 模块设计：
2. 控制器设计
3. 基本描述：

控制单元基于指令的opcode字段（Instr31：26）和funct字段（Instr5：0）计算控制信号。

1. 模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Op [5:0] | I | 用于识别指令的功能 |
| Func [5:0] | I | 用于辅助op来识别指令 |
| RegDst | O | 控制写入端地址选择  0：寄存器堆写入端地址选择Rt字段  1：寄存器堆写入端地址选择Rd字段 |
| RegWrite | O | GRF的写使能信号  0：无效  1：把数据写入寄存器堆中对应寄存器 |
| ALUSrc | O | 控制ALU的操作  0：ALU输入端B选择寄存器堆输出R[rt]  1：ALU输入端B选择extend(immediate) |
| MemWrite | O | DM的写使能信号  0：无效  1：数据存储器DM写数据（输入） |
| MemtoReg | O | 控制数据从ALU读出还是从DM读出  0：寄存器堆写入端数据来自ALU输出  1：寄存器堆写入端数据来自DM输出 |
| Branch | O | beq分支指令的控制信号 |
| ExtOp [1:0] | O | EXT功能的选择信号  00：符号扩展  01：零扩展  10：加载至高位  11：符号扩展之后，左移两位 |
| ALUControl [3:0] | O | ALU功能的选择信号  0000：加法运算  0001：减法运算  0010：或运算 |

1. 支持指令集：
   1. addu 指令：
   2. 功能：无符号加法，不考虑溢出
   3. 操作：GPR[rd] 🡨 GPR[rs] + GPR[rt]
   4. 编码：000000[31:26] rs[25:11] rt[20:16] rd[15:11] 00000[10:6] 100001[5:0]
   5. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 1 | 1 | 0 | 0 | 0 | 0 | xx | 0000 |

* 1. subu 指令：
  2. 功能：无符号减法，不考虑溢出
  3. 操作：GPR[rd] 🡨 GPR[rs] - GPR[rt]
  4. 编码：000000[31:26] rs[25:11] rt[20:16] rd[15:11] 00000[10:6] 100010[5:0]
  5. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 1 | 1 | 0 | 0 | 0 | 0 | xx | 0001 |

* 1. ori 指令：
  2. 功能：或立即数
  3. 操作：GPR[rt] 🡨 GPR[rs] OR zero\_extend(immediate)
  4. 编码：001101[31:26] rs[25:11] rt[20:16] immediate[15:0]
  5. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 0 | 1 | 1 | 0 | 0 | 0 | 01 | 0010 |

* 1. lw 指令：
     1. 功能：加载字，从内存中读取4个字节
     2. 操作：GPR[rt] 🡨 GPR[base] + sign\_extend(immediate)
     3. 编码：100011[31:26] base[25:11] rt[20:16] offset[15:0]
     4. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 0 | 1 | 1 | 0 | 1 | 0 | 00 | 0000 |

* 1. sw 指令：
     1. 功能：存储字，向内存中写入4个字节
     2. 操作：Addr 🡨 GPR[base] + sign\_extend(immediate)

Memory[Addr] 🡨 GPR[rt]

* + 1. 编码：101011[31:26] base[25:11] rt[20:16] offset[15:0]
    2. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| x | 0 | 1 | 1 | 0 | 0 | 00 | 0000 |

* 1. beq 指令：
     1. 功能：当两个待比较寄存器相等时，跳转到分支地址
     2. 操作：if ( GPR[rs] == GPR[rt] )

PC 🡨 PC + 1 + sign\_extend( offset || 02 )

else

PC 🡨 PC + 1

* + 1. 编码：000100[31:26] rs[25:11] rt[20:16] offset[15:0]
    2. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| x | 0 | 0 | 0 | 0 | 1 | 00 | 0001 |

* 1. lui 指令：
     1. 功能：立即数加载至最高位
     2. 操作：GPR[rt] 🡨 GPR[zero] + immediate || 016
     3. 编码：001111[31:26] 00000[25:11] rt[20:16] immediate[15:0]
     4. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 0 | 1 | 0 | 0 | 0 | 0 | 10 | 0000 |

* 1. nop 指令：
     1. 功能：不执行任何操作
     2. 操作：无
     3. 编码：0000000000000000000000000000000 [31:0]
     4. 控制信号：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | xxxx |

1. 控制真值表：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Opcode | Funct | RegDst | RegWrite | ALUSrc | MemWrite | MemtoReg | Branch | ExtOp | ALUControl |
| addu | 000000 | 100001 | 1 | 1 | 0 | 0 | 0 | 0 | xx | 0000 |
| subu | 000000 | 100011 | 1 | 1 | 0 | 0 | 0 | 0 | xx | 0001 |
| ori | 001101 | xxxxxx | 0 | 1 | 1 | 0 | 0 | 0 | 01 | 0010 |
| lw | 100011 | 0 | 1 | 1 | 0 | 1 | 0 | 00 | 0000 |
| sw | 101011 | x | 0 | 1 | 1 | 0 | 0 | 00 | 0000 |
| beq | 000100 | x | 0 | 0 | 0 | 0 | 1 | 00 | 0001 |
| lui | 001111 | 0 | 1 | 1 | 0 | 0 | 0 | 10 | 0000 |

RegDst = addu || subu

RegWrite = addu || subu || ori || lw || lui

ALUSrc = ori || lw || sw || lui

MemWrite = sw

MemtoReg = lw

Branch = beq

ExtOp[0] = ori

ExtOp[1] = lui

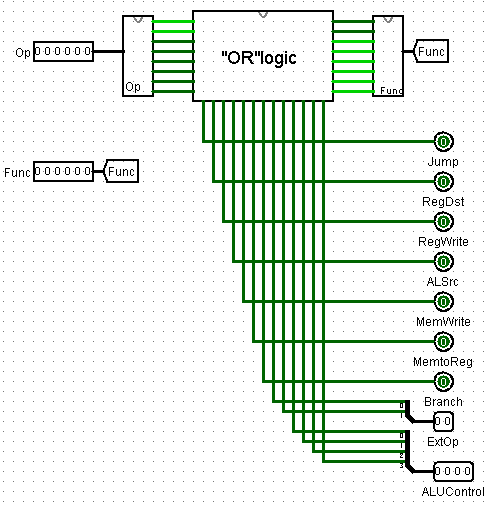
ALUControl[0] = subu || beq

ALUControl[1] = ori

ALUControl[2] = 0

ALUControl[3] = 0

1. 模块设计：



1. 测试CPU
2. 首先测试addu、subu、ori、lui、nop这四个指令

ori $1,$0,1

ori $2,$2,30

ori $3,$2,50

nop

lui $4,100

lui $5,200

lui $5,300

nop

addu $6,$1,$2

addu $7,$5,$6

addu $7,$7,$7

nop

subu $8,$7,$2

subu $9,$8,$0

subu $9,$9,$9

nop

导出的机器码为

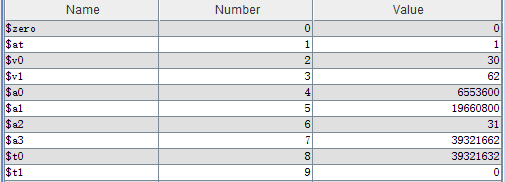
34010001 3442001e 34430032 00000000

3c040064 3c0500c8 3c05012c 00000000

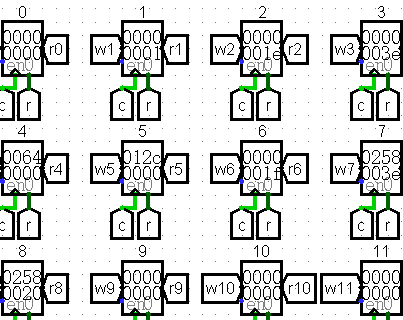
00223021 00a63821 00e73821 00000000

00e24023 01004823 01294823 00000000

参与运算的寄存器显示应为



实际结果为



1. 其次整体测试全部指令

.data

arr: .space 40

.text

ori $10,$0,4

ori $11,$0,0

ori $12,$0,3

sw $11,arr($11)

addu $11,$11,$10

sw $11,arr($11)

addu $11,$11,$10

sw $11,arr($11)

addu $11,$11,$10

beq $11,$12,loop1

lui $13,100

loop1: subu $11,$11,$10

lw $14,arr($11)

subu $11,$11,$10

lw $15,arr($11)

subu $11,$11,$10

lw $16,arr($11)

beq $11,$0,loop2

ori $17,100

loop2: ori $18,200

导出的机器码为

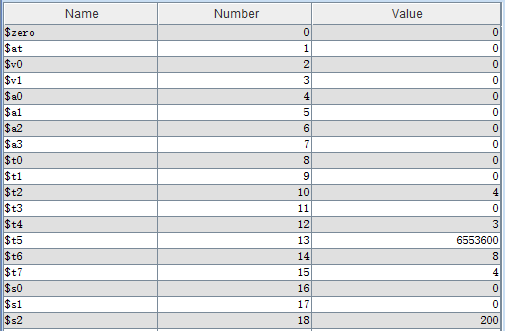
340a0004 340b0000 340c0003 ad6b0000 016a5821 ad6b0000

016a5821 ad6b0000 016a5821 116c0001 3c0d0064 016a5823

8d6e0000 016a5823 8d6f0000 016a5823 8d700000 11600001

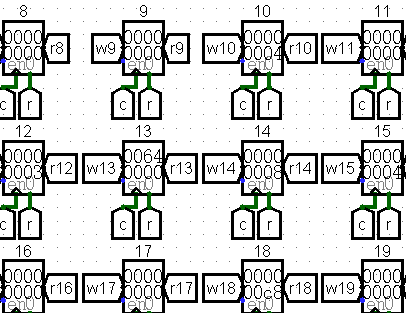
36310064 365200c8

参与运算的寄存器和内存结果应为



实际结果为





测试完成

1. 思考题
2. 若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

假设主存空间为4GB，即232B，故可使用32位的PC。优点是直接指出地址位置，缺点是耗费晶体管。

而指令字长也为32位，它们是按字对齐的，故最多存放230条指令，故可用30位的PC，取址时左移两位。优点是节省晶体管，缺点是取址时需要移位器。

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。

ROM为只读存储器，数据能够得到保护，不会被改动。符合IM在加载指令后的执行过程中不能被改动的特点。

RAM为随机访问存储器，可以进行读、写两种操作。符合DM在面对sw、lw两种操作时的读、写功能的实现，同时DM不需要特别快的速度，但需要足够的存储空间。缺点为RAM不支持跨地址操作，所以无法实现sb、lb等非4字节操作指令。

寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和地址，符合GRF的工作需要。

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 = f5f0

ALUSrc = + o5o1o0

MemtoReg = o5o0

RegWrite = f5f0 + + o5o1o0

nPC\_Sel = o5o4o3o1o0

ExtOp = o5o1o0

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 = f5f0

ALUSrc = + o5o1o0

MemtoReg = o5o0

RegWrite = f5f0 + + o5o1o0

nPC\_Sel = o5o4o3o1o0

ExtOp = o5o1o0

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。

nop指令没有对逻辑电路中的元件进行任何操作，它的存在与否对电路没有影响。如果不将nop加入控制信号真值表，则当控制器接收这条指令时，并不能将它识别为任何有效指令。

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的容量为256MB，表示数据的地址范围是0x3000\_0000—0x3FFF\_FFFF，在进行片选操作时，增加一个把接收到地址的高4位与0x3进行比较的片选信号，小于0x3存在之前的DM，大于0x3存在下一个DM，而下一个DM最大存储地址为0x0000\_0080，故不再需要片选信号。

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。

形式验证时要确定电路在哪一级电路上的测试是正确的，使用模型检验的方法看两个电路在描述上是否一致。

对组合逻辑来说，不存在状态寄存器，其输出值Z[t]不依赖于前面的输入值X[t-i](1≤i≤t)。这时只要对每个输入向量证明其输出向量相同。

对一个时序电路而言，可以把它看成一个有限状态机。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。

形式验证的优点如下：

①形式验证是对指定描述的所有可能的情况进行验证，覆盖率达到了100%。

②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。

③形式验证的验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。

形式验证的优点如下：

①不能发现代码中的功能错误和时序错误，规模太大的话验证时间更大。

②不能有效的验证电路的性能，如电路的时延和功耗等。