# Logisim单周期处理器

20373864 谭立德

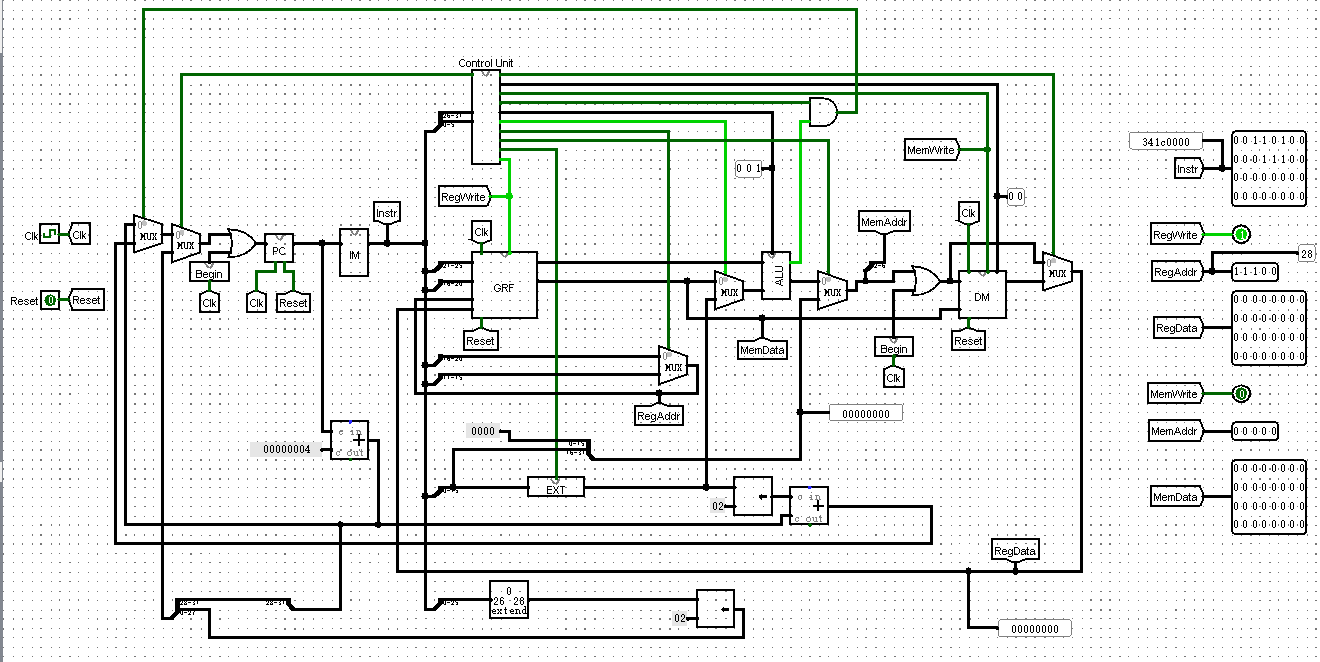


图0 顶层设计图

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为logisim实现的单周期MIPS - CPU，支持的指令集包含{ lw、sw、beq、addi、j、ori、lui、add、sub、and、or、addu、subu、lb、sb、lh、sh }。为了实现这些功能，CPU主要包含了IM、GRF、DM、ALU、PC、CU ，这些模块按照自顶向下的顶层设计逐级展开。

### （二）关键模块定义

#### 1. GRF （通用寄存器组，也称为寄存器文件、寄存器堆）

图1 GRF设计

GRF端口定义：

表0 GRF端口表

|  |  |  |
| --- | --- | --- |
| 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：不可向GEF中写入数据 |
| A1 | 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至RD1 |
| A2 | 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至RD2 |
| A3 | 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 |
| WD3 | I | 32位数据输入信号 |
| RD1 | O | 输出指定的寄存器中的32位数据 |
| RD2 | O | 输出指定的寄存器中的32位数据 |

GRF模块功能定义：

表1 GRF功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，所有寄存器储存的数值清零 |
| 2 | 读数据 | 读出A1，A2地址对应寄存器中所储存的数据到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. DM （数据存储器）：

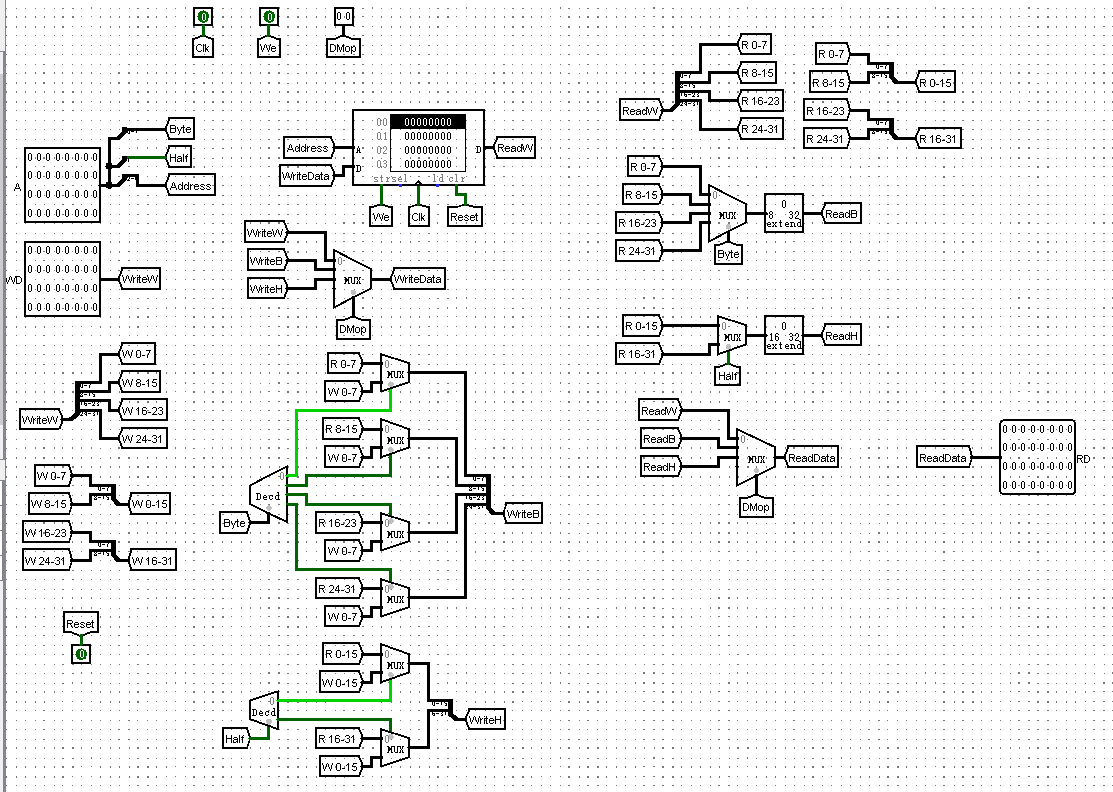


图2 DM设计

DM端口定义：

表2 DM端口表

|  |  |  |
| --- | --- | --- |
| 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DM中写入数据  0：不可向DM中写入数据 |
| A | I | 5位地址输入信号，指定中储存器上的地址，将其中存储的数据读出至RD1 |
| WD | I | 32位数据输入信号 |
| RD | O | 输出储存器指定地址上的32位数据 |

DM模块功能定义：

表3 DM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | Reset信号有效时，储存器储存的所有数值清零 |
| 2 | 读数据 | 读出A地址对应储存器中所储存的数据到RD |
| 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 |

#### ALU （算术逻辑运算单元）：

图3 ALU设计

ALU端口定义：

表4 ALU端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| SrcA | I | 32位运算数输入信号 |
| SrcB | I | 32位运算数输入信号 |
| ALU Control | I | 3位逻辑运算选择信号，选择进行哪种逻辑运算 |
| Zero | O | 输出比较两运算数比较的1位输出 |
| ALU Result | O | 输出对两运算数进行指定逻辑运算后的32位结果 |

ALU模块功能定义：

表5 ALU功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 计算 | 根据控制信号进行对应的逻辑计算并输出 |
| 2 | 比较 | 判断两个输入是否相等 |

#### IM （指令存储器）：

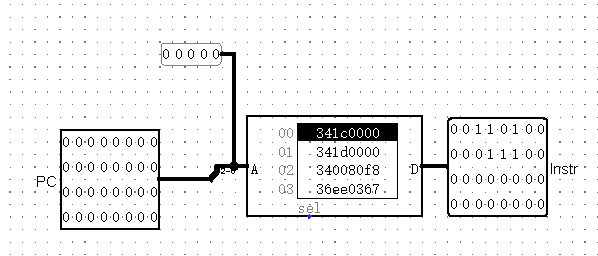


图4 IM设计

IM端口定义：

表6 IM端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC | I | 5位输入地址信号 |
| Instr | 0 | 输出地址所储存32位指令 |

IM模块功能定义：

表7 IM功能表

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读指令 | 根据输入输出对应32位指令 |

#### Control Unit （指令译码器）：

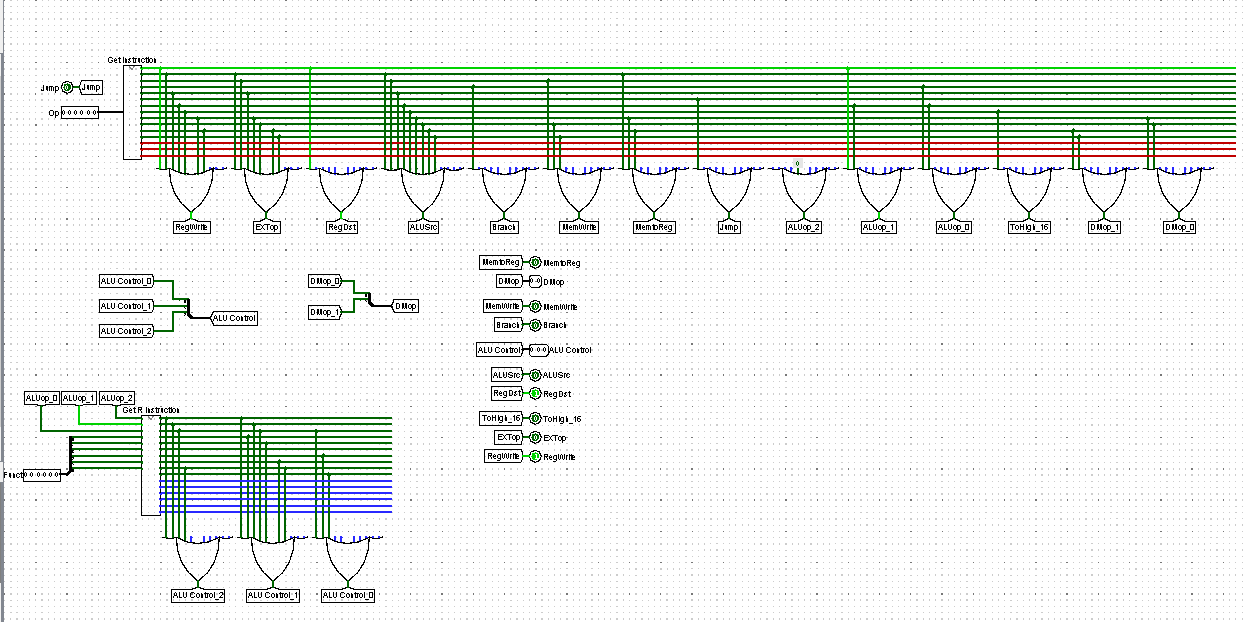


图5 Control Unit设计

Control Unit端口定义：

表8 Control Unit端口表

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Opcode[5:0] | I | 指令操作码 |
| Funct[5:0] | I | 指令功能码 |
| Jump | O | 跳转信号 |
| ToHigh16 | O | 高位置位信号 |
| ExtOp | O | 位扩展方式 |
| MemtoReg | O | 读内存信号 |
| MemWrite | O | 内存写使能信号 |
| Branch | O | 分支信号 |
| ALUCtrl[2:0] | O | ALU控制信号 |
| ALUSrc | O | ALU操作数2的来源0：寄存器1：立即数 |
| RegDst | O | 寄存器写地址选择0：Instr[20:16] 1：Instr[15:11] |
| RegWrite | O | 寄存器写使能信号 |
| DMop[1:0] | O | 存储、读取方式控制信号 |

### （三）重要机制实现方法

#### 1. J类型指令

根据输入判断和ALU模块协同工作算出跳转地址后跳转。

#### 2. R类型指令

根据输入判断和ALU模块协同工作算出结果后存储回寄存器堆中以实现指令R类型指令。

#### 3. I类型指令

根据输入判断和ALU模块和DM模块协同工作支持I类型指令。

## 测试方案

测试程序:

lui $t0,0x0004 # lui：立即数0x0004加载至t0寄存器的高位

lui $t1,0x0008 # lui：立即数0x0008加载至t1寄存器的高位

ori $t3,$zero,0x00002000 # ori：zero寄存器中的内容与立即数0x00002000进行或运算，储存在t3寄存器中

sw $t0,4($t3) # sw：把t0寄存器中值（1Word），存储到t3的值再加上偏移量4,所指向的RAM中

sw $t0,8($t3) # sw：把t0寄存器中值（1Word），存储到t3的值再加上偏移量8,所指向的RAM中

loop:add$t2,$t2,$t1 # add：t1寄存器中的值加上t2寄存器中的值后存到t2寄存器中

lw $t4,4($t3) # lw：把t3寄存器的值+4当作地址读取存储器中的值存入t4

lui $t5,0x0004 # lui：立即数0x0004加载至t5寄存器的高位

sub $t7,$t6,$t5 # sub：t6寄存器中的值减去t5寄存器中的值后存到t7寄存器中

sub $t0,$t0,$t5 # sub：t0寄存器中的值减去t5寄存器中的值后存到t0寄存器中

add $t6,$t6,$t0 # add：t6寄存器中的值加上t0后存到t6寄存器中

beq $t0,$t1,loop # beq：判断t0的值和t1的值是否相等，相等转

loop

add $t0,$t0,$t5 # add：t0寄存器中的值加上t5后存到t0寄存器中

lui $v0,0x0001 # lui：立即数0x0001加载至v0寄存器的高位lui$v1,0x0002 # lui：立即数0x0002加载至v1寄存器的高位add $v0,$v0,$v1 # add：v0寄存器中的值加上v1后存到v0寄存器中

add $v1,$v0,$v1 # add：v0寄存器中的值加上v1后存到v1寄存器中

ori $a0,$v0,0xffff # ori：v0寄存器中的内容与立即数0xffff进行或运算，储存在a0寄存器中

sub $a1,$a0,0x0000ffff # sub：a0寄存器中的值减去立即数0x0000ffff后存到a1寄存器中

loop2: sub $a2,$v1,$v0 # sub：v1寄存器中的值减去v0中的值后存到a2寄存器中

add $a1,$a2,$a1 # add：a2寄存器中的值加上a1后存到a1寄存器中

beq $a1,$v1,loop2 # beq：判断a1的值和v1的值是否相等，相等转loop2

对应机器码：3c080004 3c090008 340b2000 ad680004 01495020 8d6c0004

3c0d0004 01cd7822 010d4020 01c87020 1109fff9 010d4020 3c020001

3c030002 00431020 00431820 3444ffff 3c010000 3421ffff 00812822

00623022 00c52820 10a3fffd

## 三、思考题

### （一）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。

解：

合理，IM使用ROM存储指令保证了指令在程序运行期间不会改变。DM使用RAM则同时具备读、写和复位功能，可以存放、修改数据，支持sw和lw指令。GRF使用Register临时存储单元来实现临时存储功能，此功能与实际的CPU中相同。缺点：RAM为按字节存储，因此无法直接实现lb，sb等指令。

### 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。

解：

nop不加入真值表，即nop在Control Unit中不能被识别为任何有效指令，r任何控制信号均为零，CPU不进行任何操作，等效于空指令。

### 上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 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。

解：

在本系统中不需要修改，因为在本系统中，地址为5位地址，起始地址0x0000和0x3000均被截断高位，起始地址即均为0。对于偏移量不能跨地址的情况，可以把所有的address全部右移两位即可正常运行。

如果超过地址限制，可以用多个存储器联合，2^n个存储器采用n位片选信号来表示地址的高位即可。

### 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。

解：

形式验证的优点如下：

(1)形式验证对指定描述的所有可能的情况进行了验证

(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。

形式验证的缺点如下：

1. 验证方法复杂抽象，难以准确把握。
2. 形式验证是数学逻辑分析而不是电路分析，不能有效的验证电路的性能。附1：

主译码器真值表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | Opcode | Reg  Write | ExtOp | RegDst | ToHigh16 | ALUSrc | Branch | Mem  Write | MemtoReg | ALUop | Jump | DM  op | … |
| R | 000000 | 1 | x | 1 | 0 | 0 | 0 | 0 | 0 | 010 | 0 | xx |  |
| lw | 100011 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 000 | 0 | 00 |  |
| sw | 101011 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 000 | 0 | 00 |  |
| beq | 000100 | 0 | 1 | x | 0 | 0 | 1 | 0 | x | 001 | 0 | xx |  |
| addi | 001000 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 000 | 0 | xx |  |
| j | 000010 | 0 | x | x | 0 | x | x | 0 | x | xxx | 1 | xx |  |
| ori | 001101 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 011 | 0 | xx |  |
| lui | 001111 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | xxx | 0 | xx |  |
| lb | 100000 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 000 | 0 | 01 |  |
| sb | 101000 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 000 | 0 | 01 |  |
| lh | 100001 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 000 | 0 | 10 |  |
| sh | 101001 | 0 | 1 | x | 0 | 1 | 0 | 1 | x | 000 | 0 | 10 |  |
| … |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

附2：

ALU译码器真值表：

|  |  |  |
| --- | --- | --- |
| ALUop | Funct | ALUControl |
| 000 | X | 010（加） |
| 001 | X | 110（减） |
| 010 | 100000(add) | 010（加） |
| 010 | 100010(sub) | 110（减） |
| 010 | 100100(and) | 000（与） |
| 010 | 100101(or) | 001（或） |
| 010 | 101010(slt) | 111（小于置位） |
| 011 | X | 001（或） |
| … |  |  |