# Logisim 单周期处理器

20373864　　谭立德

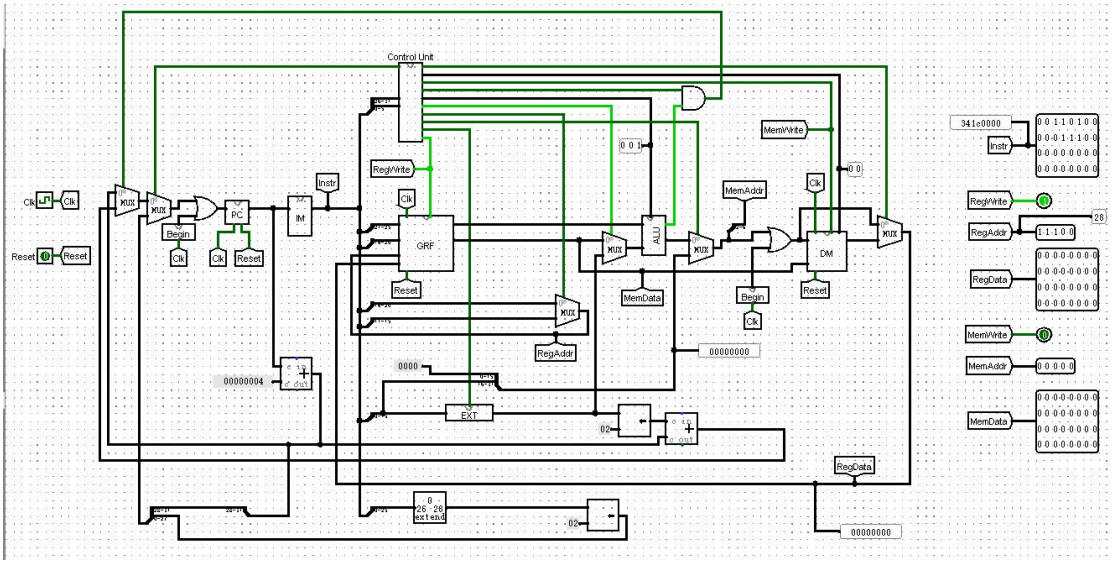


图 0　顶层设计图

## 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）

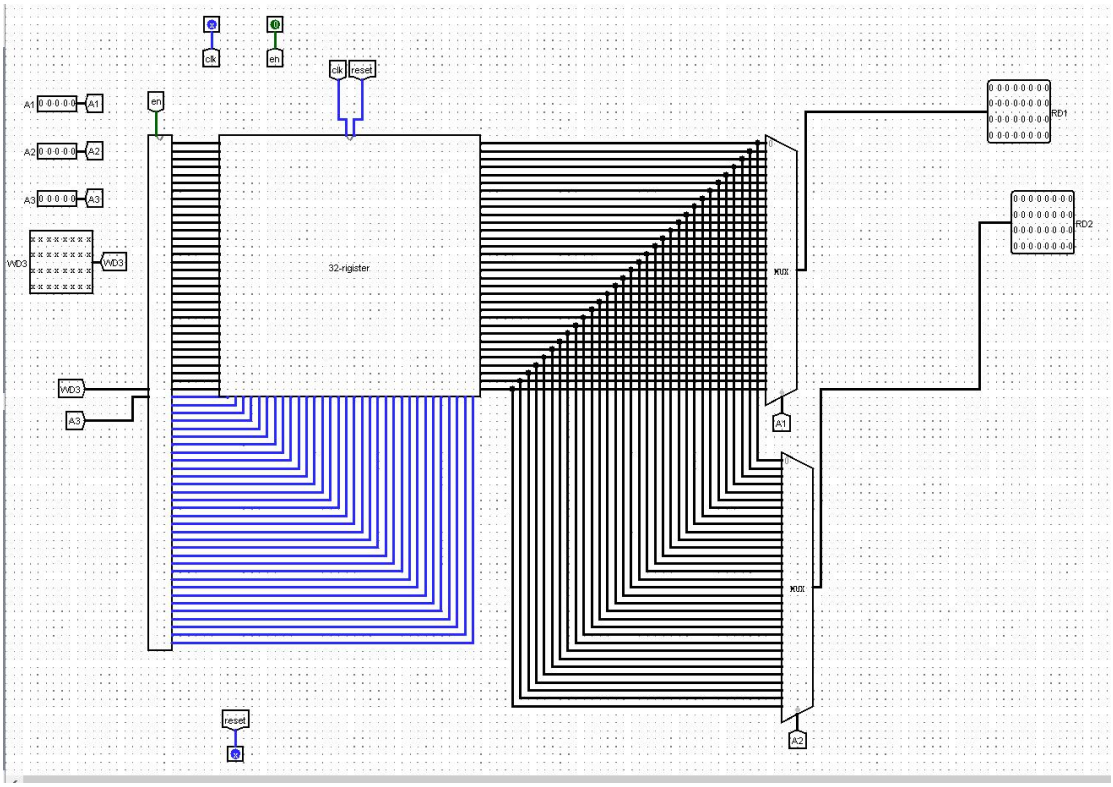


图 1 GRF 设计

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<br>1：复位<br>0：无效 |
| We | I | 写使能信号<br>1：可向 GRF 中写入数据<br>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 |

2. DM （数据存储器）：

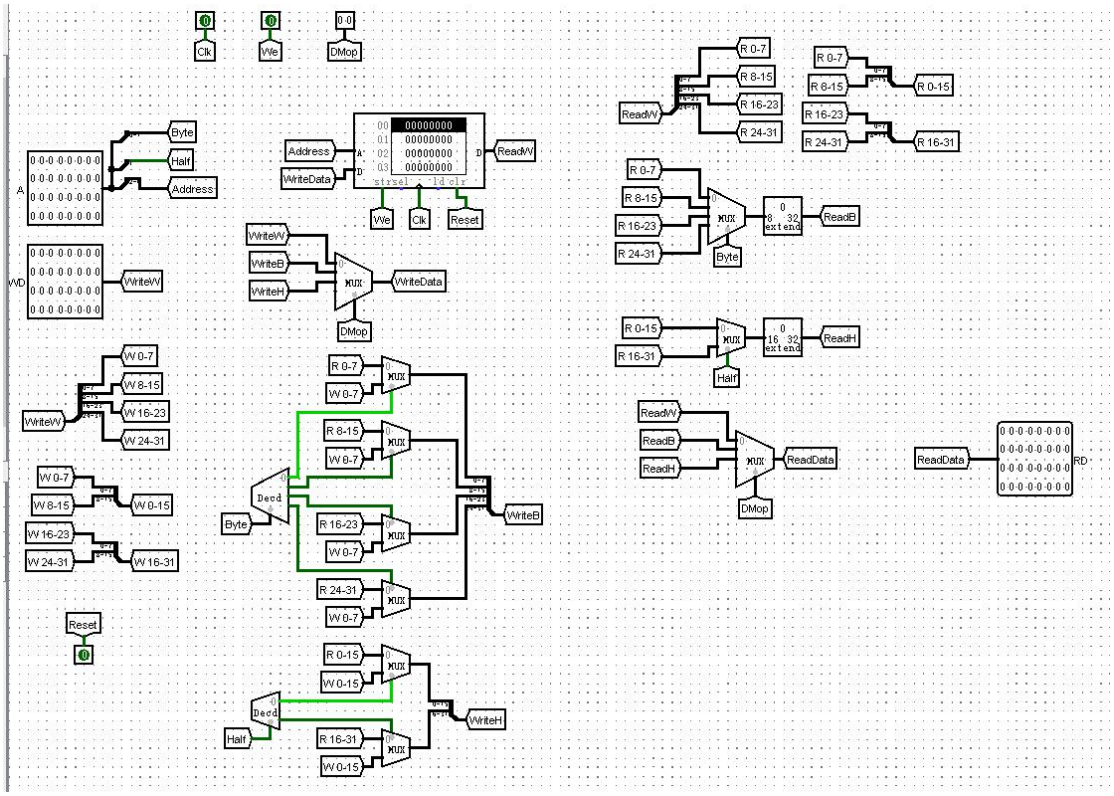


图 2 DM 设计

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<br>1：复位<br>0：无效 |
| We | I | 写使能信号<br>1：可向 DM 中写入数据<br>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 |

3. ALU （算术逻辑运算单元）：

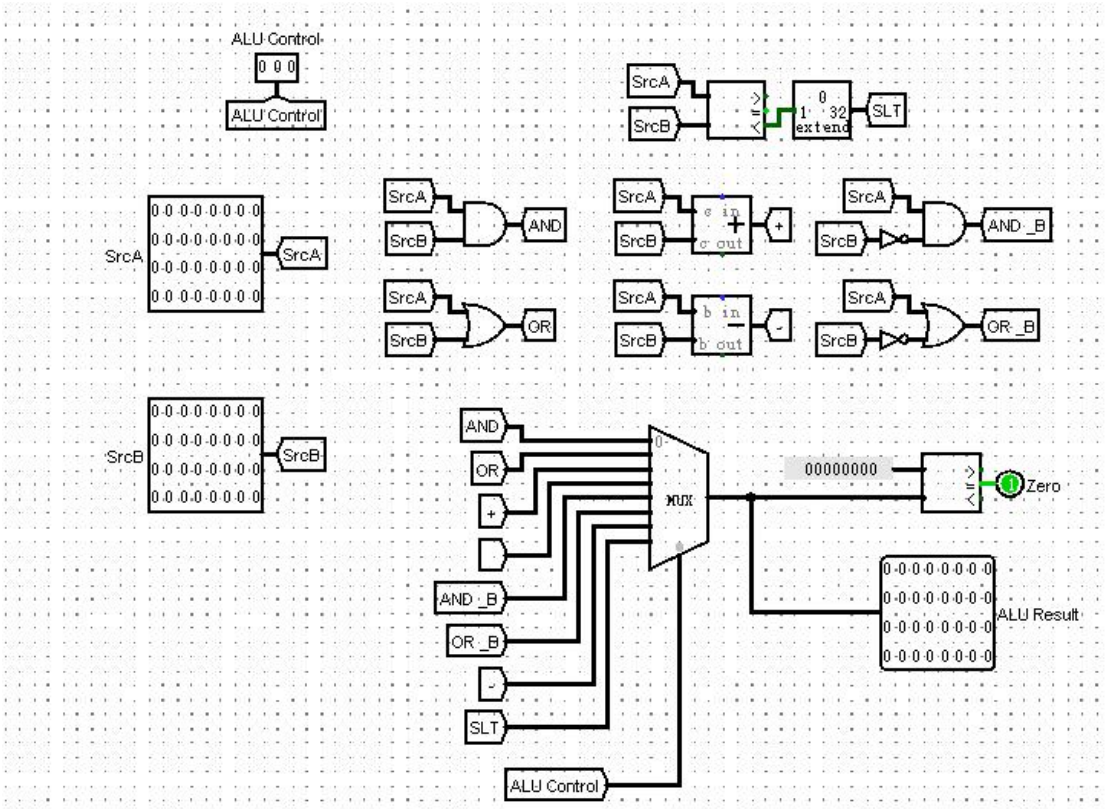


图 3 ALU 设计

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 |

4. IM （指令存储器）：

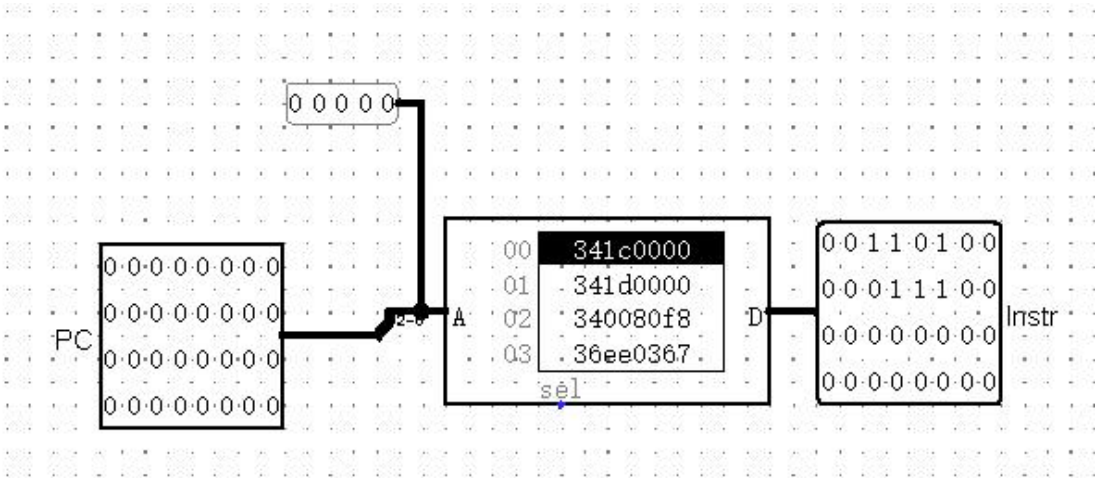


图 4 IM 设计

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 |

5. Control Unit （指令译码器）：

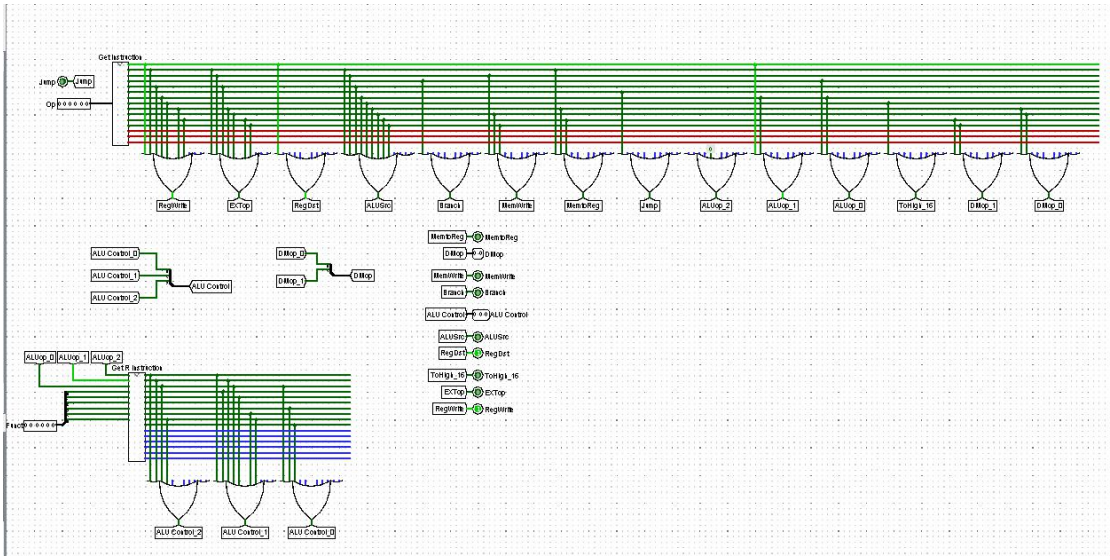

图 5 Control Unit 设计

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。

## 二、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。

### （二）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。

**（三）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。

**（四）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 | Ext Op | RegDst | ToHigh 16 | ALUSrc | Branch | Mem Write | MemtoReg | ALU op | 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） |
| … | | |