Logisim单周期CPU设计文档

一、模块

1.IFU

（1）介绍

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。

（2）端口定义

表格 1 IFU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| ifJ | I | 1 | 当前指令是否为J |
| ifBcom | I | 1 | 当前指令是否满足跳转要求 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| clk | I | 1 | 时钟信号 |
| Instr | O | 32 | 当前指令 |

（3）功能定义

表格 2 IFU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 | 当时钟上升沿到来，根据PC[6:2]从IM取指令 |
| 3 | 计算下一个PC | 如果当前指令满足跳转要求，PC=PC+4+EXT(imm||02)  如果当前指令为J指令，PC=PC[31:28]||instr\_index||02  否则，PC=PC+4 |

2.GRF

（1）介绍

通用寄存器组，也称为寄存器文件、寄存器堆。可以存取32位数据。

（2）端口定义

表格 3 GRF端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A1 | I | 5 | 指定32个寄存器中的一个，输出其中数据到RD1 |
| A2 | I | 5 | 指定32个寄存器中的一个，输出其中数据到RD2 |
| A3 | I | 5 | 指定32个寄存器中的一个，写入Data数据 |
| Data | I | 32 | 输入数据 |
| WE | I | 1 | 写入使能信号 1: 可写入 0: 不可写入 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| clk | I | 1 | 时钟信号 |
| RD1 | O | 32 | A1指定寄存器中的数据 |
| RD2 | O | 32 | A2指定寄存器中的数据 |

（3）功能定义

表格 4 GRF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，所有寄存器的数据清零 |
| 2 | 读数据 | 读出A1,A2指定寄存器中数据到RD1,RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿时，将Data写入A3指定寄存器 |

3.ALU

（1）介绍

算术逻辑单元，提供32位加、减、与、或运算，不检测溢出。

（2）端口定义

表格 5 ALU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A | I | 32 | ALU的输入1 |
| B | I | 32 | ALU的输入2 |
| sel | I | 2 | 选择信号，00: A+B 01: A-B 10: A&B 11: A|B |
| C | O | 32 | 运算结果 |
| notzero | O | 1 | 判断C是否为零1: C=0 0: C!=0 |

（3）功能定义

表格 6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 运算 | 按照sel信号选择C为A和B做什么运算得到的结果 |
| 2 | 判零 | 判断C是否为0 |

4.DM

（1）介绍

存储数据。

（2）端口定义

表格 7 DM端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| address | I | 5 | 待操作地址 |
| data | I | 32 | 待输入数据 |
| WE | I | 1 | 写入使能信号1: 可写入 0: 不可写入 |
| clk | I | 1 | 时钟信号 |
| reset | I | 1 | 异步复位信号 1: 复位 0: 无效 |
| out | O | 32 | 读出的数据 |

（3）功能定义

表格 8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 写数据 | 当WE有效且时钟上升沿时，将data写入address地址 |
| 2 | 读数据 | 从address中读取数据，输出至out |
| 3 | 复位 | 当复位信号有效时，所有ROM的数据清零 |

5.EXT

（1）介绍

将16位立即数扩展为32位。

（2）端口定义

表格 9 EXT端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| A | I | 16 | 待扩展的16位立即数 |
| sel | I | 2 | 扩展方式选择信号  00: 无符号扩展  01: 有符号扩展  10: 后面拼接两个0后符号扩展  11: 加载至高位 |
| B | O | 32 | 扩展后的数 |

（3）功能定义

表格 10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 扩展 | 按照sel信号选择B为A做什么扩展得到的结果 |

6.Controller

（1）介绍

根据指令有关信息（opcode，func）判断指令类型，进而得到各个选择器、使能信号等的数据，决定各组件控制信号。

（2）端口定义

表格 11 Controller端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口 | 输入输出 | 位数 | 描述 |
| Instr | I | 32 | 指令内容 |
| regSlt | O | 1 | 选择进入GRF的B的来源 0: rt 1: rd |
| regWE | O | 1 | GRF的写入使能信号 |
| dmWE | O | 1 | DM的写入使能信号 |
| extOp | O | 2 | EXT的选择信号 |
| aluOp | O | 2 | ALU的选择信号 |
| aluB | O | 1 | ALU的B端口来源 0: rt 1: 扩展后的立即数 |
| toReg | O | 1 | 存入寄存器数据来源 0: DM数据 1: ALU结果 |
| ifBeq | O | 1 | 判断是否为Beq指令 |
| rs | O | 5 | 读rs寄存器序号 |
| rt | O | 5 | 读rt寄存器序号 |
| rd | O | 5 | 读rd寄存器序号 |
| imm | O | 16 | 读立即数 |
| jcom | O | 1 | 判断是否为J指令 |

（3）真值表（instr略）（非严格 无x）

表格 12 Controller真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 端口 | addu | subu | ori | lw | sw | lui | beq | j |
| Instr  （略） | 000000  100001 | 000000  100011 | 001101 | 100011 | 101011 | 001111 | 000100 | 000010 |
| regSlt | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| regWE | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| dmWE | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| extOp0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| extOp1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| aluOp0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| aluOp1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| aluB | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| toReg | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| ifBeq | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| jcom | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

二、测试程序

（1）代码

.data

arr:.space 40

.text

ori $t0,0

ori $s0,10

loop:

beq $t0,$s0,loop\_out

subu $t1,$t1,$t1

subu $t4,$t4,$t4

lj:

beq $t1,$s0,ljout

lw $t3,arr($t4)

addu $t4,$t4,4

addu $t3,$t3,$t1

sw $t3,arr($t4)

addu $t1,$t1,1

j lj

ljout:

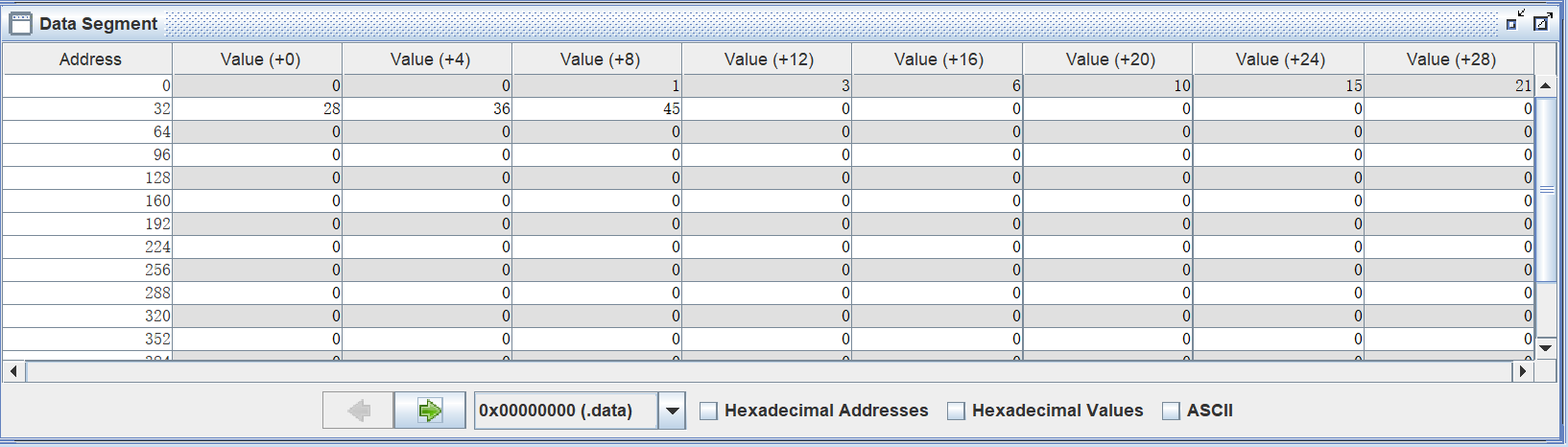
addu $t0,$t0,1

j loop

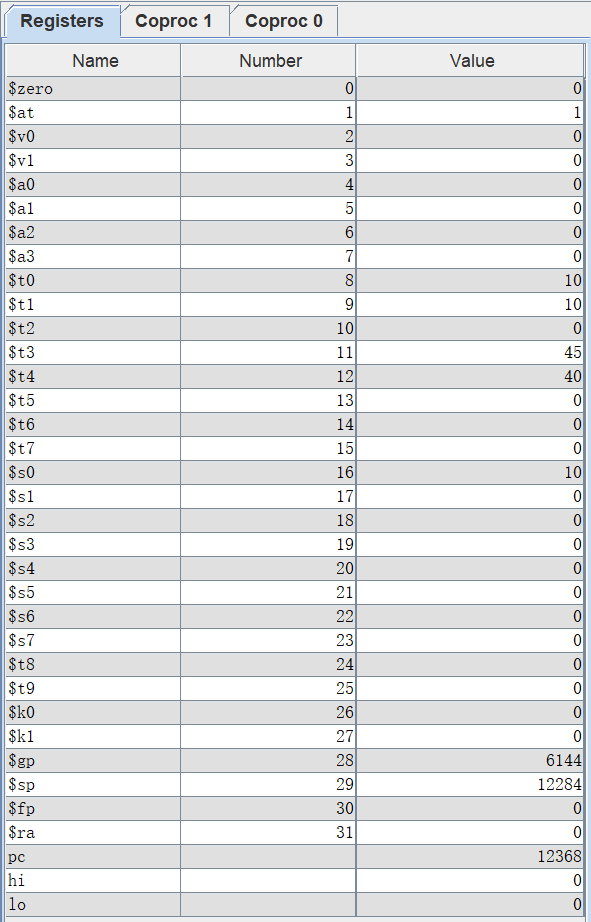
loop\_out:

beq $t2,$t2,loop\_out

（2）MARS输出结果：

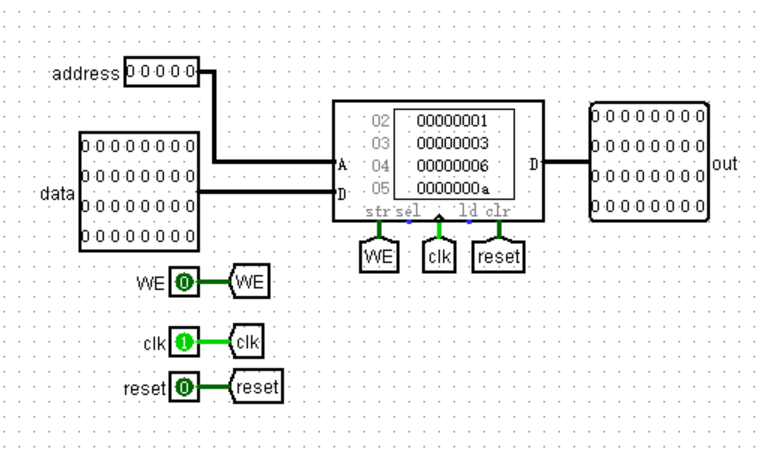


图表 1 MARS数据输出结果

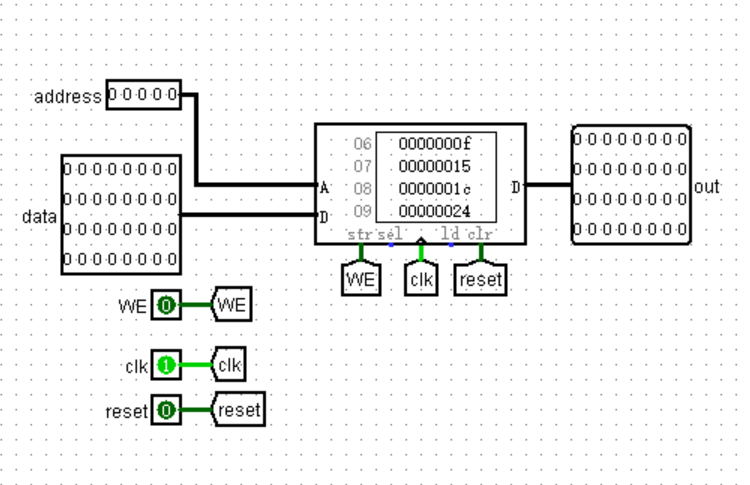


图表 2 MARS寄存器运行结果

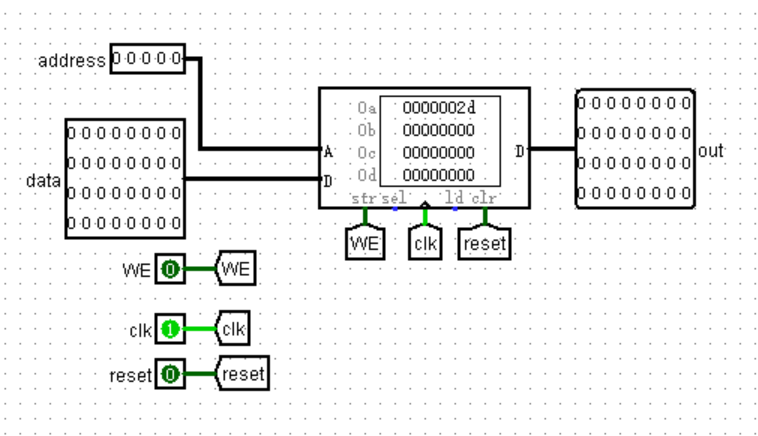
（2）CPU运行结果：



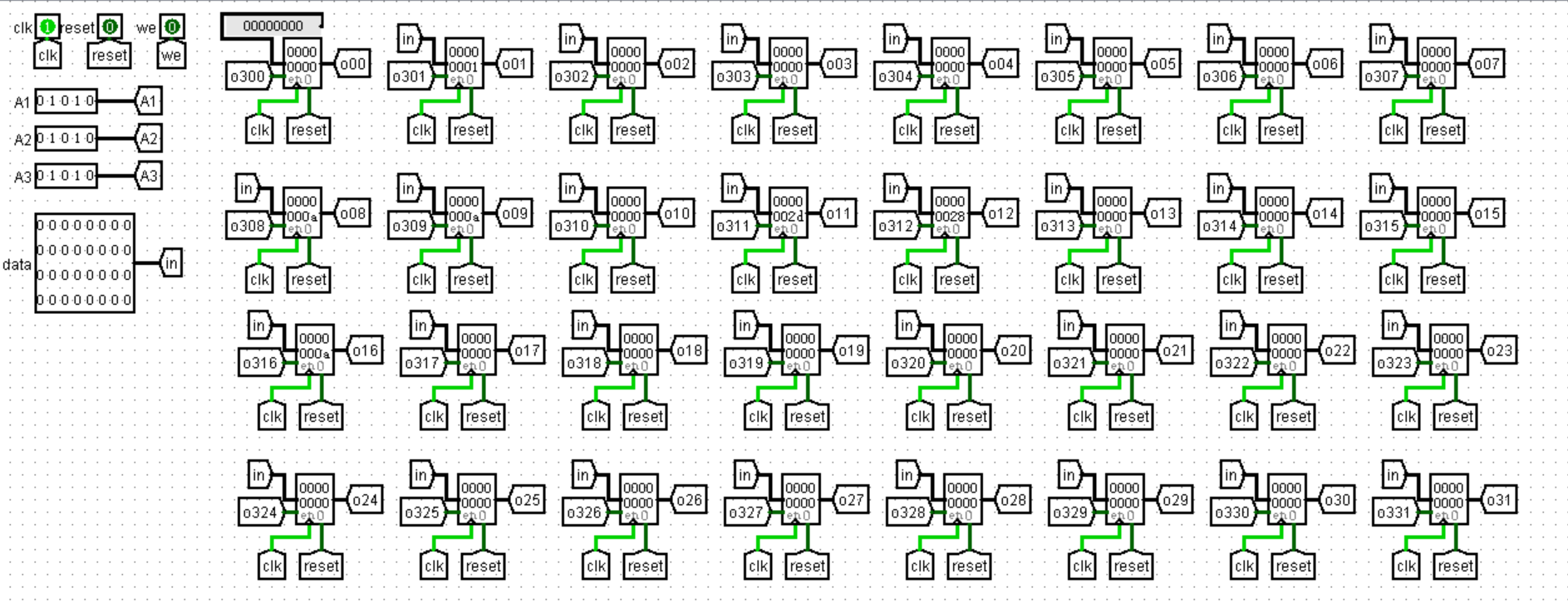
图表 3 CPU数据运行结果1



图表 4 CPU数据运行结果2



图表 5 CPU数据运行结果3



图表 6 CPU寄存器运行结果

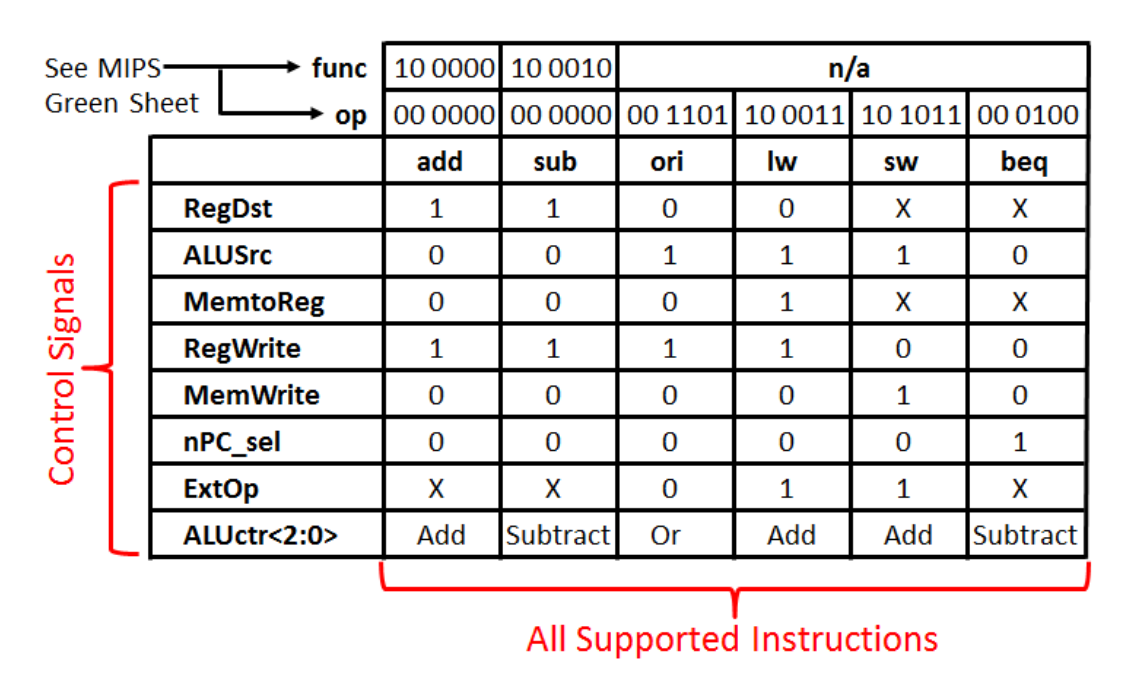
符合预期。

三、思考题

1. 若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。  
   优点：直接取低5位作为地址；缺点：实现jr等方法时会比较麻烦，与MIPS实现有较大差别，测试较困难
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。IM只需读入，DM需要读写清零，GRF寄存器堆使用寄存器。

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。）



图表 7 参考图表

RegDst=func5~fun4~fun3~fun2~fun1~fun0~op5~o4~op3~op2~op1~op0||func5~fun4~fun3~fun2fun1~fun0~op5~o4~op3~op2~op1~op0

其余类似。

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。

与上文中实现一样，看做0。略。

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。

不会进行操作，使能信号等不会有效，对电路无影响。

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。

暴力减法取地址。

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。

优点：严谨，能保证电路正确性

缺点：麻烦，复杂