# P3-Logisim单周期实验报告

## 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，处理器为32位处理器，支持的指令集包含{ addu, subu, ori, lw, sw, beq, lui, nop,j}。为了实现这些功能，CPU主要包含了Controller（控制器）、IFU（取指令单元）、GRF（通用寄存器组，也称为寄存器文件、寄存器堆）、ALU（算术逻辑单元）、DM（数据存储器）、 EXT（位扩展器）等基本部件，这些模块分成3级。

### （二）关键模块定义

#### 1. GRF

（可使用表格进行端口说明）

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| reset | I | 复位信号 |
| we | I | 可写入信号 |
| A1 | I | 读出地址1 |
| A2 | I | 读出地址2 |
| A3 | I | 写入地址 |
| WD | I | 写入数据 |
| RD1 | O | 读出数据1 |
| RD2 | O | 读出数据2 |

#### 2. DM

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| reset | I | 复位信号 |
| MemWrite | I | 可写入信号 |
| MemRead | I | 可读出信号 |
| A | I | 地址 |
| D1 | I | 写入数据 |
| D2 | O | 读出数据 |

#### 3. ALU

该模块包括add,sub,or,and子模块，用于进行运算

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| In1 | I | 运算数1 |
| In2 | I | 运算数2 |
| OP | I | 操作信号 |
| Zero | I | 运算结果是否为0 |
| Out | I | 运算结果 |

运算符对应表(四位操信号备用)

|  |  |
| --- | --- |
| 0000 | 加 |
| 0001 | 减 |
| 0010 | 或 |
| 0011 | 与 |
| 0100 | 将in2左移16位（用于lui） |

#### 4. IFU

该模块保证每个时钟周期PC根据信号决定自增4或进行beq/q的跳转

并且持续输出读取到的指令

该模块包括PC\_add用于自增4，sel26用于获取2~6位信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| reset | I | 复位信号 |
| 26bits | I | 26位立即数用于beq |
| PCSrc | I | 是否beq |
| PCJump | I | 是否j |
| Out | O | 指令 |

5. EXT

用于拓展信号16->32

0：无符号拓展 1：有符号拓展

6. sel26

用于获取2~6位信号

### （三）控制器Controller定义

该模块对指令的op和func进行解码，输出对应的信号。

利用和逻辑和或逻辑识别指令然后生成信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op | I | 指令的6位op部分 |
| func | I | 指令的6位func部分 |
| RegDst | O | 寄存器写入选择信号  0：使用rt写入  1：使用rd写入 |
| RegWrite | O | 寄存器写入信号 |
| ALUSrc | O | ALU运算数选择信号  0：grf  1：imm |
| branch | O | PC变更选择信号  0：自增4  1：beq |
| PCJump | O | PC变更选择信号  0：branch  1：j |
| MemRead | O | 内存写信号 |
| MemWrite | O | 内存读信号 |
| MemtoReg | O | 写入寄存器信号  0：ALU结果  1：内存 |

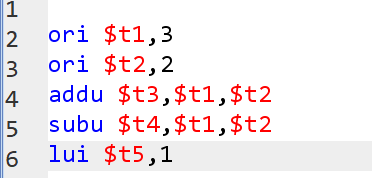
指令信号对应表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | RegDst | RegWrite | ALUSrc | branch | PCJump | MemRead | MemWrite | MemtoReg | extsel | ALU |
| addu | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | x | 0000 |
| subu | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | x | 0001 |
| ori | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0010 |
| lw | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0000 |
| sw | x | 0 | 1 | 0 | 0 | 0 | 1 | x | 1 | 0000 |
| beq | x | 0 | 0 | 1 | 1 | 0 | 0 | x | x | 0001 |
| lui | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0100 |
| j | x | 0 | x | 1 | 0 | 0 | 0 | x | x | x |
| nop | x | x | x | x | x | x | x | x | x | x |

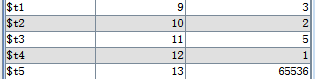
## 二、测试方案

### （一）典型测试样例

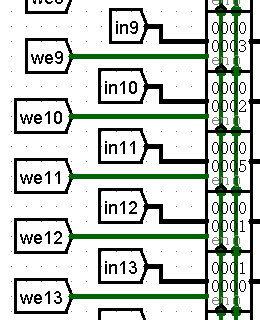
#### 1. ALU功能测试



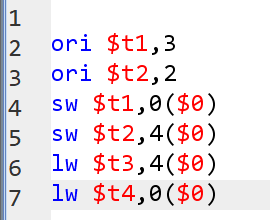
期望值：



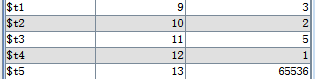
实际值：



#### 2. DM功能测试

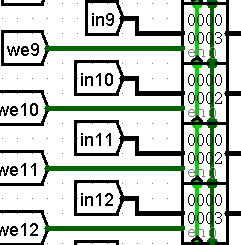


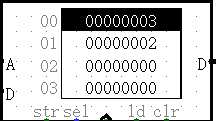
期望值：



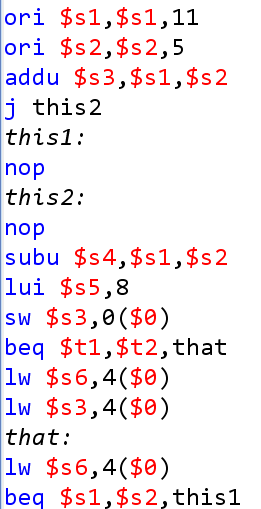


实际值：

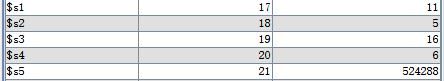




#### 3. 其他指令综合测试

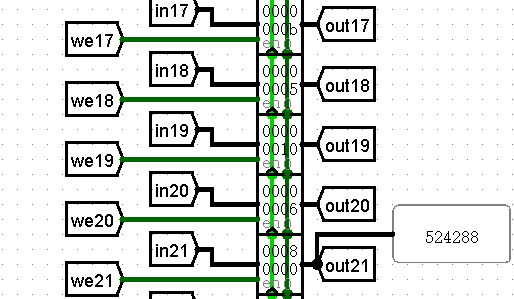


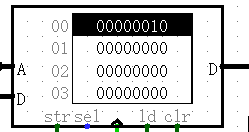
期望值：





实际值：

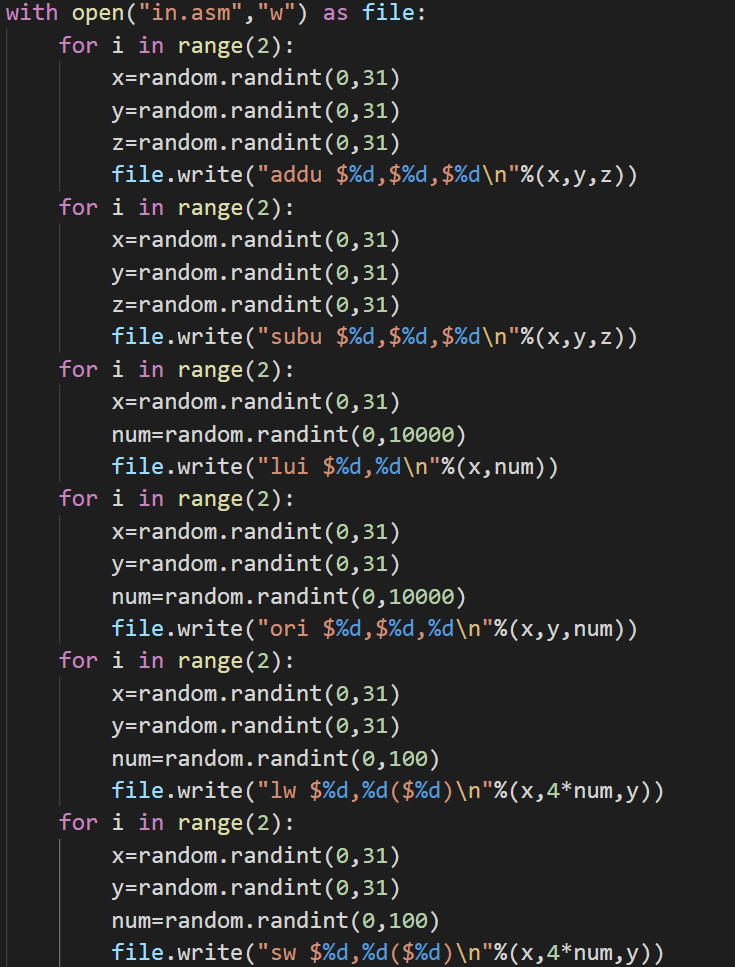




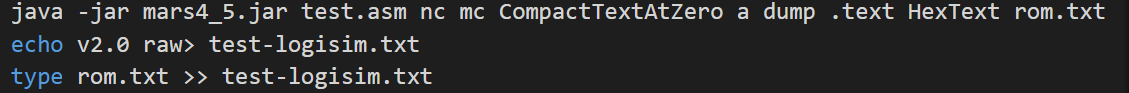
### （二）自动测试工具

#### 1. 测试样例生成器

|  |
| --- |
| import os  import random  import re  with open("in.asm","w") as file:      for i in range(2):          x=random.randint(0,31)          y=random.randint(0,31)          z=random.randint(0,31)          file.write("addu $%d,$%d,$%d\n"%(x,y,z))      for i in range(2):          x=random.randint(0,31)          y=random.randint(0,31)          z=random.randint(0,31)          file.write("subu $%d,$%d,$%d\n"%(x,y,z))      for i in range(2):          x=random.randint(0,31)          num=random.randint(0,10000)          file.write("lui $%d,%d\n"%(x,num))      for i in range(2):          x=random.randint(0,31)          y=random.randint(0,31)          num=random.randint(0,10000)          file.write("ori $%d,$%d,%d\n"%(x,y,num))      for i in range(2):          x=random.randint(0,31)          y=random.randint(0,31)          num=random.randint(0,100)          file.write("lw $%d,%d($%d)\n"%(x,4\*num,y))      for i in range(2):          x=random.randint(0,31)          y=random.randint(0,31)          num=random.randint(0,100)          file.write("sw $%d,%d($%d)\n"%(x,4\*num,y))      # for i in range(2):      #     x=random.randint(0,31)      #     y=random.randint(0,31)      #     num=random.randint(0,100)      #     file.write("beq $%d,$%d,%d\n"%(x,y,4\*num)) |



#### 2. 自动执行脚本



|  |
| --- |
| os.system("java -jar mars4\_5.jar in.asm nc mc CompactTextAtZero a dump .text HexText rom.txt")  with open("rom.txt","r") as file:      content=file.read()  with open("CPU.circ","r") as file:      circmy=file.read()  circmy=re.sub(r'addr/data: 5 32([\s\S]\*)</a>',"addr/data: 5 32\n"+content+"</a>",circmy)  with open("CPU.circ","w") as file:      circmy=file.write(circmy) |

#### 3. 正确性判定脚本

没有想到合适的提取GRF数值的方案，采取直接对比的方法。

## 三、思考题

1. 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。

我认为合理。IM中存只读的指令，因此用ROM，DM需要读取和存储，因此使用RAM，GRF为寄存器堆，应使用Register，可存可取。

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指令机器码为全0，此时由控制器信号全部为0，决定了DM和Register都是不可存入的状态，仅PC会自增4，CPU在本周期等价于无操作。

1. 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。

MARS中PC高位为0x3…可以将0x3全部修改为0x0，然而本项目中不会涉及到高位，仅需取出2~6位用作片选信号即可。我选择设置了sel26模块用于选取ALU和PC中的2-6位作为DM和IM的地址。

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，简要阐述相比于测试，形式验证的优劣之处。

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。在集成电路设计中，形式验证是一种集成电路设计的验证方法，它的主要思想是通过使用形式证明的方式来验证一个设计的功能是否正确。形式验证可以分为三大类：抽象解释（Abstract Interpretation）、形式模型检查（Formal Model Checking，也被称作特性检查）和定理证明（Theory Prover）。

优：通过数学的方式系统性的检查电路，易于发现细小的漏洞。

劣：过程较为复杂，需要较多的知识背景。