# 计算机组成原理P3单周期CPU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，支持的指令集包含{addu, subu, and, or, sll(nop), sllv, slt, jr, j, jal, beq, ori, lui, addi, lw, sw, lh, sw, lb, sb}。为了实现这些功能，CPU主要包含了PC、NPC、IM、GRF、ALU、DM、EXT、Controller。

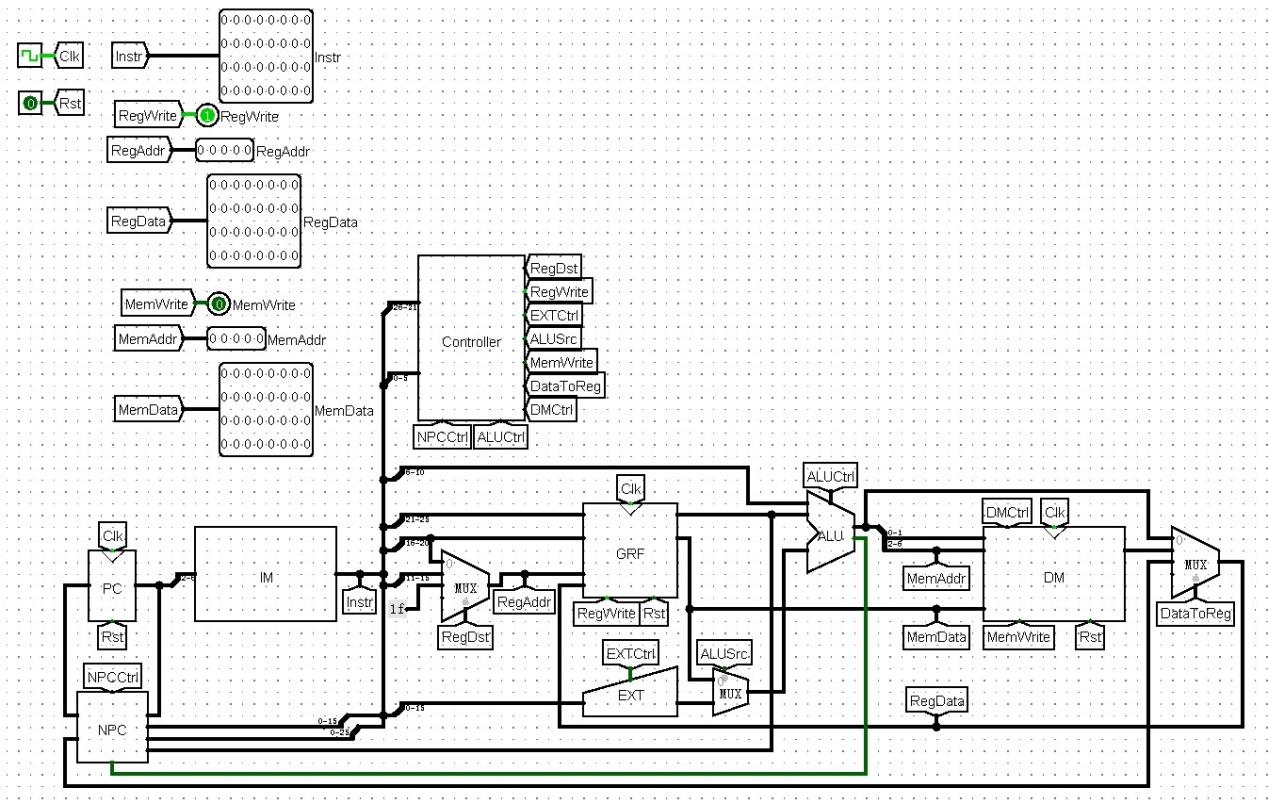
模块间的协同关系主要为：PC存储当前指令地址，NPC计算下一条指令地址、IM存储所有指令、GRF为32个通用寄存器、ALU执行主要计算、DM存储数据到内存中、EXT对立即数进行位扩展、Controller计算控制信号。除Controller之外的模块组成数据通路，Controller则控制数据通路中数据的选择、计算等。

图1 CPU整体电路

### （二）关键模块定义

### 1. PC（Program Counter）

端口定义：

表1 PC端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | NextPC | I | 32 | 下一条指令的地址 |
| 2 | Clk | I | 1 | 时钟信号 |
| 3 | Rst | I | 1 | 异步复位信号 |
| 4 | PC | O | 32 | 当前指令地址 |

内部逻辑说明：

用一个32位寄存器存储当前指令地址，当时钟上升沿到来且使能信号为1时将新的指令地址写入寄存器。当异步复位信号有效时就将寄存器复位为起始地址0x00000000。

### 2. NPC（Next PC）

端口定义：

表2 NPC端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | PC | I | 32 | 当前指令地址 |
| 2 | Offset | I | 16 | branch跳转的偏移量 |
| 3 | Index | I | 26 | j或jar跳转地址的2-27位 |
| 4 | Reg | I | 32 | jr指令的跳转地址 |
| 5 | Brflag | I | 1 | branch跳转条件是否满足 |
| 6 | Ctrl | I | 3 | 选择下条指令的地址 |
| 7 | PCAdd4 | O | 32 | 当前指令地址加4 |
| 8 | NextPC | O | 32 | 下一条指令地址 |

内部逻辑说明：

分别计算出PC+4、branch跳转地址、j或jal跳转地址、jr跳转地址，然后根据Brflag和Ctrl信号选择要输出的下条指令地址NextPC。PCAdd4用于jal指令将PC+4的值存入$ra寄存器。

### 3. IM（Instruction Memory）

端口定义：

表3 IM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | A | I | 5 | 所取指令的地址 |
| 2 | RD | O | 32 | 32位指令 |

内部逻辑说明：

用ROM存储所有指令。根据地址取出相应指令

### 4. GRF（General Register File）

端口定义：

表4 GRF端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | Clk | I | 1 | 时钟信号 |
| 2 | Rst | I | 1 | 异步复位信号，将寄存器中的值全部清零 |
| 3 | WE | I | 1 | 写使能信号 |
| 4 | A1 | I | 5 | 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 |
| 5 | A2 | I | 5 | 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RD2 |
| 6 | A3 | I | 5 | 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 |
| 7 | WD | I | 32 | 32位数据输入信号 |
| 8 | RD1 | O | 32 | 输出A1指定的寄存器中的32位数据 |
| 9 | RD2 | O | 32 | 输出A2指定的寄存器中的32位数据 |

内部逻辑说明：

通过多路分解器进行输入数据WD和使能信号WE的分配，A3为选择信号；通过多路选择器进行输出数据RD1和RD2的选择，A1和A2分别为选择信号。

### 5. ALU（Arithmetic Logical Unit）

端口定义：

表5 ALU端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | SrcA | I | 32 | 操作数A |
| 2 | SrcB | I | 32 | 操作数B |
| 3 | Shamt | I | 5 | 移位位数 |
| 4 | Ctrl | I | 4 | 控制信号，选择一种计算结果输出 |
| 5 | Out | O | 32 | 计算结果 |
| 6 | Flag | O | 1 | 输出逻辑运算的结果，用于branch类指令 |

内部逻辑说明：

支持加、减、与、或、异或、移位、比较等运算。分开进行每种计算，最后通过多路选择器和Ctrl信号来选择输出结果。

### 6. DM（Data Memory）

端口定义：

表6 DM端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | A | I | 5 | 5位地址输入信号，将数据写入该地址 |
| 2 | ByteSel | I | 2 | 用于对半字和字节进行读写 |
| 3 | Ctrl | I | 2 | 控制信号，选择对字、半字还是字节操作 |
| 4 | WE | I | 1 | 写使能信号 |
| 5 | Clk | I | 1 | 时钟信号 |
| 6 | Rst | I | 1 | 异步复位信号，将内存清零 |
| 7 | WD | I | 32 | 32位数据输入信号 |
| 8 | RD | O | 32 | 输出地址A存储的数据 |

内部逻辑说明：

用RAM存储数据，对地址A中的数据进行读写。以半字、字节为单位进行存入的时候，要注意和同一字地址中其他字节进行拼接，读取的时候注意进行位扩展。

### 7. EXT（Extender）

端口定义：

表7 EXT端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | In16 | I | 16 | 16位立即数输入信号 |
| 2 | Ctrl | I | 1 | 控制信号，选择符号扩展还是零扩展 |
| 3 | Out32 | I | 32 | 32位数据输出信号 |

内部逻辑说明：

将16位立即数进行零扩展或符号扩展成32位后输出，使用Bit Extender实现。

### 8.Controller

端口定义：

表8 Controller端口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 位数 | 描述 |
| 1 | Op | I | 5 | 指令中的Op字段 |
| 2 | Funct | I | 2 | R型指令中的funct字段 |
| 3 | RegDst | O | 2 | GRF的写入地址选择信号 |
| 4 | RegWrite | O | 1 | GRF写使能信号 |
| 5 | EXTCtrl | O | 1 | EXT控制信号 |
| 6 | ALUSrc | O | 1 | ALU操作数选择信号 |
| 7 | MemWrite | O | 1 | DM写使能信号 |
| 8 | DataToReg | O | 2 | GRF的写入数据选择信号 |
| 9 | DMCtrl | O | 2 | DM控制信号 |
| 10 | NPCCtrl | O | 3 | NPC控制信号 |
| 11 | ALUCtrl | O | 4 | ALU控制信号 |

内部逻辑说明：

通过与逻辑来实现指令的识别，再通过或逻辑实现从指令到控制信号的映射。为了简化识别电路，可以先将R指令识别出来，之后对R指令再单独进行识别。

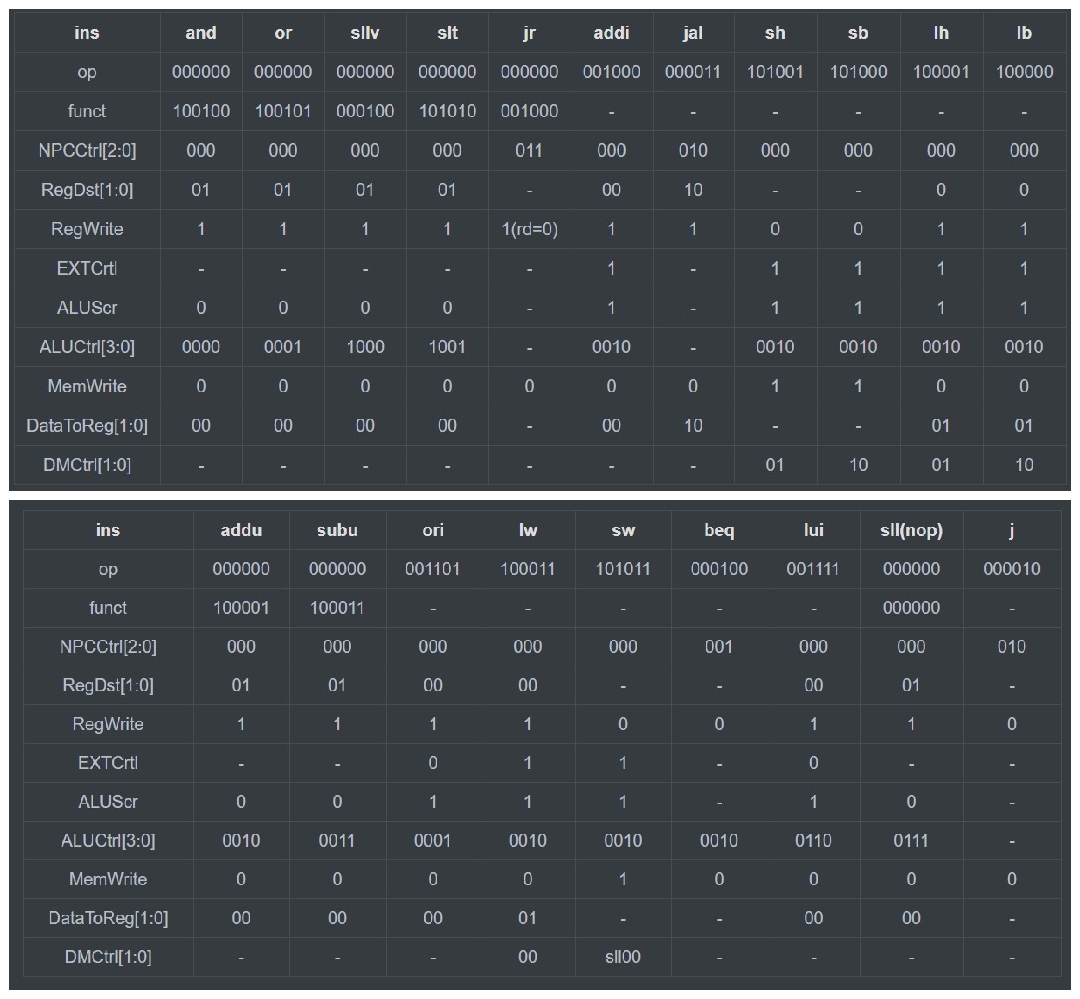


图2 指令与控制信号对照表

### （三）重要机制实现方法

#### 1. 跳转

NPC模块和ALU模块协同工作支持指令beq的跳转机制。

NPC模块内置了判定单元和计算单元来独立支持指令j、jal、jr 的跳转机制。

#### 2. 半字、字节存取

通过两位的控制信号DMCtrl来判断是对字、半字还是字节进行操作。由于RAM是以字编址，在对半字、字节读入时，需要将无关的数据与要写入的数据进行拼接，再整体写入RAM。读取时则取出整个字，选出某个位置的半字或字节进行扩展后再输出。

#### 3. 控制信号

通过与逻辑来识别指令，对于R型指令，先通过Op识别出该类型是否为R型指令，再通过Funct具体识别是哪一条R型指令。通过或逻辑产生控制信号，当一条指令需要的控制信号为1时，将其连到或门上。对于多位的控制信号，分别对每一位用一个或门控制。

## 二、测试方案

### （一）典型测试样例

#### 1. 计算指令测试

ori $a0, $0, 0

ori $a1, $0, 1

ori $a2, $0, 65535

ori $a3, $a0, 123

lui $a0, 65535

lui $a1, 0

lui $a2, 2

lui $a3, 456

addu $t0, $a0, $a1

subu $t1, $a1, $a2

sll $t2, $a3, 2

sllv $t3, $a3, $a1

slt $t4, $a0, $a1

#### 2. 跳转指令测试

ori $t0, $0, 0

ori $s0, $0, 5

for\_begin:

beq $t0, $s0, for\_end

jal func

addi $t0, $t0, 1

j for\_begin

for\_end:

beq $0, $0, end

func:

jr $ra

end:

#### 3. 存取指令测试

ori $a0, 1

ori $a1, 2

ori $a2, 3

ori $a3, 4

sw $a0, 0($0)

sw $a1, 0($a3)

sh $a2, 8($0)

sh $a3, 10($0)

sb $a0, 12($0)

sb $a1, 13($0)

sb $a2, 14($0)

sb $a3, 15($0)

lw $t0, 0($0)

lw $t1, 0($a3)

lh $t2, 8($0)

lh $t3, 10($0)

lb $t4, 12($0)

lb $t5, 13($0)

lb $t8, 14($0)

lb $t7, 15($0)

### （二）自动测试工具

#### 1. 测试样例生成器

代码来源于讨论区

#### ins_gnrt

图3 测试样例生成器部分代码

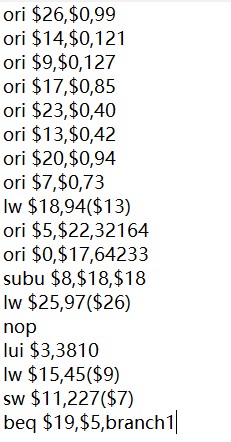


图4 生成的测试样例

#### 自动执行脚本

通过python和命令行进行自动化测试，可以实现mips代码自动导出、IM指令导入、logisim自动运行并将记录的结果格式化，以便和mips执行的结果进行比对。

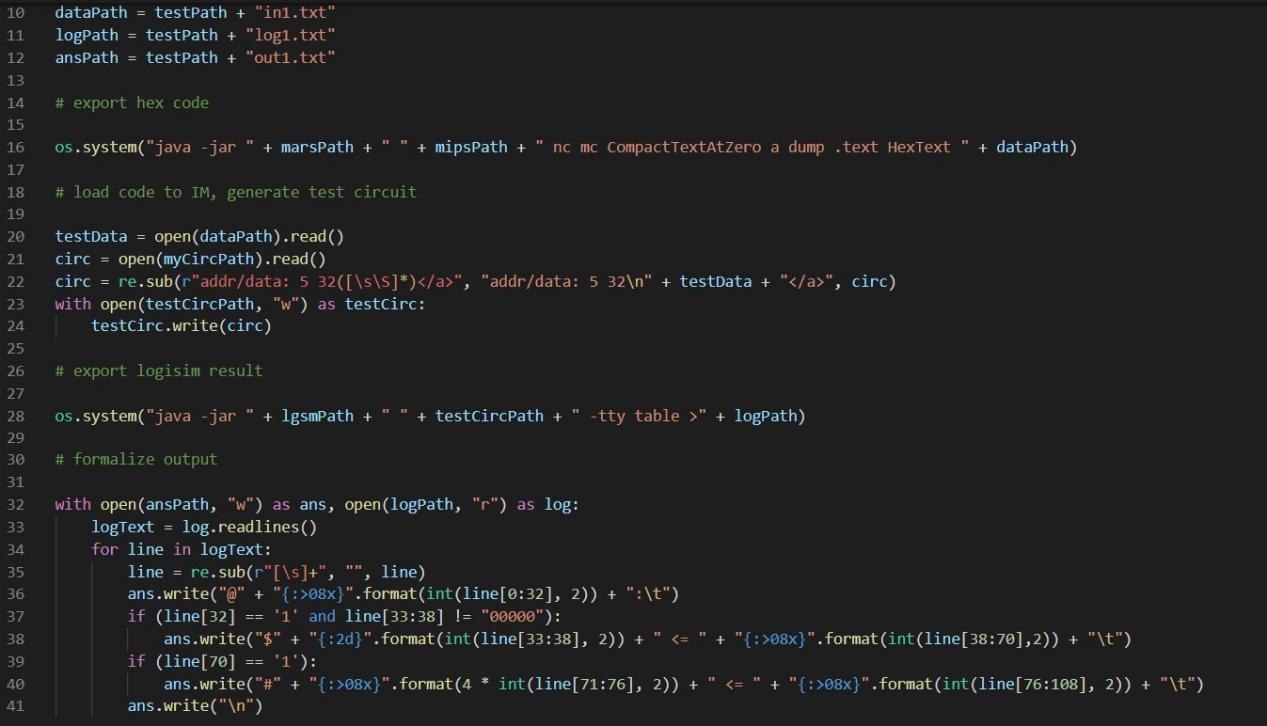


图5 自动执行脚本部分代码

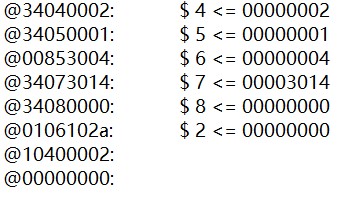


图6 格式化后的输出结果

## 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。

目前来看是合理的。因为一段指令在执行过程中不应该发生改变，所以指令存储器可用只读的ROM；而数据存储器需要同时实现读写功能，并且只根据地址进行读写，所以适合用RAM；对于GRF，需要实现一个端口的写入和两个端口的读取，并且要求0寄存器的值始终为0，所以用32个独立的寄存器来实现较好。

但在实际的系统中，指令存储器和数据存储器相互独立是不现实的。大多数计算机有一块单独的大容量内存来存储指令和数据，并且支持读和写操作。

### （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。

空指令nop实际上是指令sll $0, $0, 0，机器码的32位全为0。如果CPU实现了sll指令，那么nop会将0寄存器中的值左移0位后写入0寄存器，不会造成任何影响。

### （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。

logisim中的RAM地址最高支持24位，当访问的地址超过24位的时候，可以使用多片RAM，低24位作为RAM的地址，高位连接到片选信号用于选择哪一片RAM。

但在我实际的设计中，因为mips导出的数据是从0x00000000开始存放的而指令是从0x00003000开始存放的，两者起始地址的2-7位均是0，而由于电路中的IM和DM地址均只有5位，不会造成影响，所以暂未特殊处理这个问题。如果之后需要扩大地址范围，对于PC来说，可以将地址减去0x00003000；对于DM来说，可以利用片选信号。

### （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。

优势：可以对指定描述的所有可能的情况进行验证，覆盖率达到了百分之百；利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，不需要使用仿真测试平台和激励；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。

劣势：对一套系统建立数学模型是有很高的时间成本的，当前产品迭代很快，有了bug也可以很快修复，如果对每个版本都去进行形式验证会很费时间。其次，形式验证目前不能有效验证电路的性能，如电路的时延和功耗。