# 计算机组成原理实验报告参考模板

## 一、CPU设计方案综述

### （一）总体设计概述

使用Logisim开发一个简单的单周期CPU，总体概述如下：

1. 此CPU为32位CPU
2. 此CPU为单周期设计
3. 此CPU支持的指令集为：

{addu, subu, ori, lw, sw, beq, lui, nop, jr, j, jal}

1. nop机器码为0x00000000
2. addu, subu不支持溢出

### （二）关键模块定义

1. IFU
2. 端口说明

表1-IFU端口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将PC值置为0x00000000  0：无效  1：复位 |
| 3 | RsignPC | I | 指令是不是R型中的跳转  0：不是  1：是 |
| 4 | IsignPC | I | 指令是不是I型中的跳转  0：不是  1：是 |
| 5 | JsignPC | I | 指令是不是J型中的跳转  0：不是  1：是 |
| 6 | Rrs [31:0] | I | R型指令中的跳转时，跳转到的rs寄存器中值 |
| 7 | Iimm [15:0] | I | I型指令中的跳转时，跳转到的立即数 |
| 8 | Jadd [31:0] | I | J型指令中的跳转时，跳转到的地址 |
| 5 | Instr [31:0] | O | 读出指令 |
| 6 | PC[31:0] | O | 当前PC |

1. 功能定义

表2-IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | reset有效时，PC置为0x00000000 |
| 2 | 更新PC的值 | RsignPC=1时，PC为Rrs  IsignPC=1时，PC为PC+4+sign\_ext(Iimm)<<2  JsignPC=1时，PC为  PC[31:28]Jadd[27:2]0[1:0]  否则PC为PC+4 |
| 3 | 输出指令 | 根据PC的值，取出IM中的指令 |
| 4 | 输出PC | 输出PC |

1. GRF
2. 端口说明

表3-GRF端口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将32个寄存器中全部清零  1：清零  0：无效 |
| 3 | WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| 4 | A1[4:0] | 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到RD1 |
| 5 | A2[4:0] | 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到RD2 |
| 6 | A3[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，作为RD的写入地址 |
| 7 | WD[31:0] | I | 32位写入数据 |
| 8 | RD1[31:0] | O | 输出A1指定的寄存器的32位数据 |
| 9 | RD2[31:0] | O | 输出A2指定的寄存器的32位数据 |

1. 功能定义

表4-GRF功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | reset为1时，将所有寄存器清零 |
| 2 | 读数据 | 将A1和A2地址对应的寄存器的值分别通过RD1和RD2读出 |
| 3 | 写数据 | 当WE为1且时钟上升沿来临时，将WD写入到A3对应的寄存器内部 |

1. ALU
2. 端口说明

表5-ALU端口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | A[31:0] | I | 参与运算的第一个数 |
| 2 | B[31:0] | I | 参与运算的第二个数 |
| 3 | Instr[31:0] | I | 指令 |
| 4 | DM\_RD[31:0] | I | DM读到的数据 |
| 3 | RsignPC | O | 指令是不是R型中的跳转  0：不是  1：是 |
| 4 | IsignPC | O | 指令是不是I型中的跳转  0：不是  1：是 |
| 5 | JsignPC | O | 指令是不是J型中的跳转  0：不是  1：是 |
| 6 | Rrs [31:0] | O | R型指令中的跳转时，跳转到的rs寄存器中值 |
| 7 | Iimm [15:0] | O | I型指令中的跳转时，跳转到的立即数 |
| 8 | Jadd [31:0] | O | J型指令中的跳转时，跳转到的地址 |
| 9 | RF\_WD[31:0] | O | 回写数据 |
| 10 | WE | O | 写使能信号  0：禁止写入  1：允许写入 |
| 11 | A[4:0] | O | 读取或写入信号地址 |
| 12 | WD[31:0] | O | 32为写入数据 |
| 13 | DM\_awake | O | 唤醒指令，当读写时激活 |
| 14 | RD[31:0] | O | 32位读出数据 |

1. DM
2. 端口说明

表7-DM端口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号  0：无效  1：内存值全部清零 |
| 3 | WE | I | 写使能信号  0：禁止写入  1：允许写入 |
| 4 | A[4:0] | I | 读取或写入信号地址 |
| 5 | WD[31:0] | I | 32为写入数据 |
| 6 | DM\_awake | I | 唤醒指令，当读写时激活 |
| 6 | RD[31:0] | O | 32位读出数据 |

1. 功能定义

表8-DM功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | 当reset为1时，DM中所有数据清零 |
| 2 | 写入数据 | 当WE及awake有效时，时钟上升沿来临时，WD中数据写入A对应的DM地址中 |
| 3 | 读出数据 | Awake激活时RD读出A对应的DM地址中的值 |

1. EXT
2. 端口说明

表9-EXT端口

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号名 | 方向 | 描述 |
| 1 | imm16[15:0] | I | 代扩展的16位信号 |
| 2 | sign? | I | 无符号或符号扩展选择信号  0：无符号扩展  1：符号扩展 |
| 3 | imm32[31:0] | O | 扩展后的32位的信号 |

1. 功能定义

表10-EXT功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 无符号扩展 | 当sign?为0时，将imm16无符号扩展输出 |
| 2 | 符号扩展 | 当sign?为1时，将imm16符号扩展输出 |

### （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x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y的跳转机制。

## 二、测试方案

### （一）典型测试样例

1. 测试代码：

.text

ori $t1, $v0, 100 # t1: 100

ori $t2, $v0, 250 # t2: 250

ori $t3, $v0, 200 # t3: 200

addu $t4, $t1, $t3

# t4 = t1 + t3 (t4 = 300)

subu $t5, $t4, $t2

# t5 = t4 - t2 (t5 = 50)

sw $t5, 4($v0)

lw $t6, 4($v0)

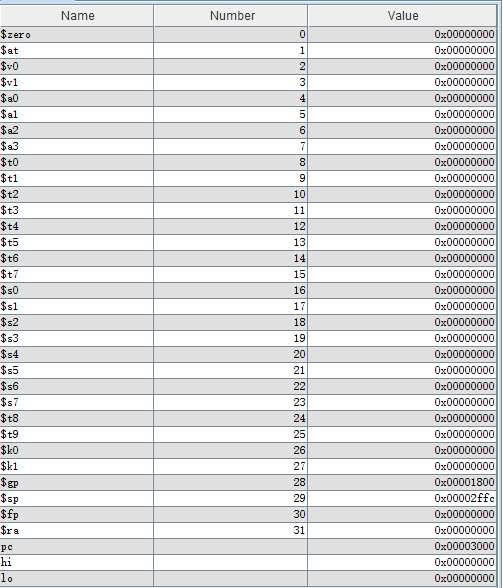
Beq:

lui $t7, 100

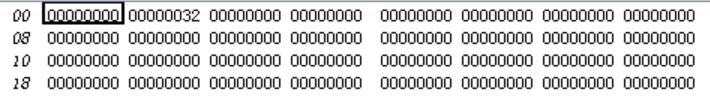
beq $t5, $t1, Beq

beq $t6, $t5, Beq

1. MARS中运行结果



1. 该CPU运行结果



## 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。

合理。

IM只需被读取，ROM只有读取功能；不需要进行写入

DM既要进行读取，又要进行写入，但是一个周期只会进行读取和写入之一，RAM的单一地址和各一个的读写端口满足了这种要求，如果使用寄存器阵列，则是一种浪费

GRF需要读写，且其与ALU直接连接，需要高速地读写，故使用寄存器堆搭建合理。

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，整个CPU只执行PC<=PC+4指令，空指令加入无异于

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。

类似MARS寄存器中存储的DM的地址被映射在0x3000\_0000到0x3fff\_ffff间，而我们的DM起始地址是0，那么，我们可以将输入地址直接减去0x3000\_0000，再作为DM的地址输入。

假如我们不确定寄存器中的存储的DM地址的起始值，我们可以将其与0x3000\_0000比较，得到片选信号。

因为P3搭建的cpu为冯诺依曼体系，理论上DM,IM是不分离的，所以Mars里面是合理的。P3简化为哈佛体系罢了

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（FormalVerification)”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。

形式验证的优点如下：

1. 所有可能的情况进行验证，覆盖率达到了100%。
2. 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. 形式验证只能检验电路设计的正确性，却无法检验其它方面如电路能耗等的优劣。