# 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

本CPU为Verilog实现的流水线MIPS - CPU，支持的指令集包含{ LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、 SUB、SUBU、MULT、MULTU、DIV、DIVU、SLL、SRL、SRA、SLLV、 SRLV、SRAV、AND、OR、XOR、NOR、ADDI、ADDIU、ANDI、ORI、 XORI、LUI、SLT、SLTI、SLTIU、SLTU、BEQ、BNE、BLEZ、BGTZ、 BLTZ、BGEZ、J、JAL、JALR、JR、MFHI、MFLO、MTHI、MTLO }为了实现这些功能，CPU主要包含了DM、GRF、乘除模块、ALU、NPC、Controller，这些模块按照由上至下、由简入繁的顶层设计逐级展开。

### （二）关键模块定义

#### 1. GRF

表1 GRF端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| W\_RegWrite | I | 写使能信号  1：可向GRF中写入数据  0：不可向GRF中写入数据 |
| Reset | I | 同步复位信号  1：复位信号有效  0：复位信号无效 |
| W\_DMout | I[31:0] | W级流水线寄存器输出的值 |
| D\_Code | I[31:0] | 当前D级的机器码 |
| W\_A3 | I[4:0] | 写入寄存器的编号 |
| RD1 | O[31:0] | 输出code[25:21]号寄存器的数据 |
| RD2 | O[31:0] | 输出code[20:16]号寄存器的数据 |
| G\_A3 | O[4:0] | 第五级流水线写入寄存器的地址 |
| G\_RegWrite | O | 第五级流水线寄存器写使能信号 |
| G\_out | O[31:0] | 第五级流水线写入寄存器的信号 |

#### 2. DM

表2 DM端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| M\_Pc | I[31:0] | 当前指令的地址 |
| CLK | I | 时钟信号 |
| Reset | I | 同步复位信号  1：有效  0：无效 |
| M\_MemWrite | I[3:0] | 写使能信号  0001:可向一个字节写入数据  0011：可向半字写入数据  1111：可向一个字写入数据  0000：不可写入数据 |
| M\_RD2 | I[31:0] | M级的rt号寄存器中的数值 |
| M\_Aluout | I[31:0] | M级Alu部件的运算结果 |
| DM\_rt\_trans | I[1:0] | DM所需rt寄存器结果的转发信号  0：无需转发  10：转发第五级流水线的结果  1：转发W级的结果 |
| W\_MemtoReg | I | 选择W级需要被转发的数据  1：W\_DMout  其他：M\_Aluout |
| W\_Aluout | I[31:0] | W级Alu的运算结果 |
| G\_out | I[31:0] | 第五级流水线写入寄存器的结果 |
| m\_data\_addr, | O[31:0] | 数据存储器待写入地址 |
| m\_data\_wdata, | O[31:0] | 数据存储器待写入数据 |
| m\_data\_byteen, | O[3:0] | 字节使能信号 |
| m\_inst\_addr | O[31:0] | M 级 PC |
| DMout | O[31:0] | DM中读出的数据 |

#### 3. 乘除模块

表3 乘除模块端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Start | I[2:0] | 001：执行mult指令  010：执行multu指令  011：执行div指令  100：执行divu指令 |
| P1 | I[31:0] | 第一个运算数 |
| P2 | I[31:0] | 第二个运算数 |
| MemtorReg | I[1:0] | 10：读出HI寄存器  11：读出LO寄存器 |
| MD | I[1:0] | 写使能信号  00：无法写入  10：向HI写入  11：向LO写入 |
| Busy | O | 0：乘除模块没有在运算  1：乘除模块正在运算 |
| MDout | O[31:0] | 从读出HI或LO寄存器读出的数据 |

#### 4. Risk\_Controler

表4 RiskController端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| E\_T\_new | I[1:0] | E级产生新结果所需周期 |
| M\_T\_new | I[1:0] | M级产生新结果所需周期 |
| W\_T\_new | I[1:0] | T级产生新结果所需周期 |
| T\_Alu\_rs\_use | I[1:0] | Alu的rs寄存器使用时间 |
| T\_Alu\_rt\_use | I[1:0] | Alu的rt寄存器使用时间 |
| T\_DM\_use | I[1:0] | DM的rt寄存器使用时间 |
| M\_A3 | I[4:0] | M级写入寄存器的编号 |
| W\_A3 | I[4:0] | W级写入寄存器的编号 |
| E\_A3 | I[4:0] | E级写入寄存器的编号 |
| G\_A3 | I[4:0] | 第五级流水线寄存器写入寄存器的编号 |
| E\_RegWrite | I | E级寄存器写使能信号 |
| M\_RegWrite | I | M级寄存器写使能信号 |
| W\_RegWrite | I | W级寄存器写使能信号 |
| G\_RegWrite | I | 第五级寄存器写使能信号 |
| Branch | I[2:0] | 跳转信号 |
| D\_code | I[31:0] | D级指令的机器码 |
| Alu\_rs | I[4:0] | Alu所需rs寄存器的编号 |
| Alu\_rt | I[4:0] | Alu所需rt寄存器的编号 |
| DM\_rt | I[4:0] | DM所需rt寄存器的编号 |
| Zero\_rs\_trans | O[1:0] | Zero部件rs寄存器的转发信号  0：无需转发  1：转发M级数据  10：转发W级数据 |
| Zero\_rt\_trans | O[1:0] | Zero部件rt寄存器的转发信号  0：无需转发  1：转发M级数据  10：转发W级数据 |
| Alu\_rs\_trans | O[1:0] | Alu部件rs寄存器的转发信号  0：无需转发  1：转发M级数据  10：转发W级数据 |
| Alu\_rt\_trans | O[1:0] | Alu部件rt寄存器的转发信号  0：无需转发  1：转发M级数据  10：转发W级数据 |
| DM\_rt\_trans | O[1:0] | DM部件rt寄存器的转发信号  0：无需转发  1：转发W级数据  10：转发第五级数据 |
| pasue | O | 暂停信号  1：暂停信号有效  0：暂停信号无效 |
| Start | I[1:0] | 0：乘除模块没有在执行运算  其他：乘除模块正在执行运算 |
| busy | I | 0：乘除模块没有在执行运算  1：乘除模块正在执行运算 |

### （三）重要机制实现方法

#### 1. 跳转

将比较器前移至D级，控制器译码出信号后，比较器便能在该级计算出比较结果，如两寄存器是否相等、寄存器的值与0的关系，将跳转信号和比较结果传输至NPC，由NPC实现跳转的操作。

#### 2. 流水线延迟槽

在NPC部件中，当不满足跳转条件时，每周期都执行pc自增4的运算。

#### 3. 转发与暂停

转发信号由冒险控制器输出。以Alu部件的转发为例，当指令传输至E级时，向冒险控制器输出Alu所需寄存器的编号，EM流水寄存器和MW流水寄存器向冒险控制器输出写入寄存器编号以及寄存器写使能信号，由冒险控制器对编号进行比对，输出转发信号。

当指令传输至D级时，由控制器进行译码，向冒险控制器输出需求者的T\_use和需求者的寄存器编号，EM和MW流水寄存器向冒险控制器输出各自新结果产生的时间T\_new、写入寄存器编号以及寄存器写使能信号，由冒险控制器进行编号比对和时间比较，输出暂停信号。

## 二、测试方案

### （一）典型测试样例

#### 1. 转发功能测试

#### ori $at,$zero,1

#### nop

#### nop

#### nop

#### nop

#### addu $v0,$at,$v0

#### nop

#### beq $v0,$zero,branch

#### ori $ra,$zero,1

#### branch:sw $at,($zero)

#### nop

#### nop

#### nop

#### nop

#### lw $v1,($zero)

#### nop

#### nop

#### beq $v1,$zero,Branch

#### ori $ra,$zero,1

#### Branch:addu $a0,$at,$a0

#### addu $a0,$a0,$a0

#### nop

#### nop

#### nop

#### nop

#### lw $a1,($zero)

#### nop

#### addu $a1,$a1,$a1

#### nop

#### nop

#### nop

#### nop

#### lw $a2,($zero)

#### nop

#### nop

#### addu $a2,$a2,$a2

#### nop

#### nop

#### nop

#### nop

#### addu $a2,$a2,$a2

#### sw $a2,($zero)

#### nop

#### nop

#### nop

#### nop

#### addu $a2,$a2,$a2

#### nop

#### sw $a2,($zero)

#### 2. 乘除模块功能测试

Ori $a0,$zero,5

Ori $a1,$zero,21

Mult $a0,$a1

Mflo $a0

Mfhi $a1

Ori $a0,$zero,5

lui $a1,$zero,ffff

mult $a0,$a1

Mflo $a0

Mfhi $a1

Ori $a0,$zero,5

Ori $a1,$zero,21

div $a1,$a0

Mflo $a0

Mfhi $a1

Ori $a0,$zero,5

lui $a1,$zero,ffff

divu $a1,$a0

Mflo $a0

Mfhi $a1

Ori $a0,$zero,5

Ori $a1,$zero,21

Mthi $a0

Mtlo $a1

#### 3. 暂停测试

ori $at,$zero,1

beq $at,$zero,branch

ori $ra,$zero,1

branch:ori $ra,$zero,1

ori $t0,$zero,2

nop

nop

nop

nop

sw $t0,($zero)

nop

nop

nop

nop

lw $t1,($zero)

beq $t1,$zero,Branch

Branch:ori $ra,$zero,1

ori $ra,$zero,1

sw $at,($zero)

nop

nop

nop

nop

lw $v0,($zero)

addu $v0,$v0,$v0

## 三、思考题

### （一）

为什么要有单独的乘除法部件而不是直接整合进ALU？为何需要有独立的HI、LO寄存器？参照对延迟槽的理解，试解释“乘除槽”。

ALU的计算只需一个周期便可计算出结果，而乘法和除法需要的周期数更多，将乘除法部件独立于ALU之外，方便设计和实现。

HI和LO寄存器不在32个寄存器之内，用来保存乘法和除法的计算结果，不能借用32个寄存器，否则可能会出现错误。

由于乘法和除法执行的时间较长，大于一个周期，当乘除法部件在执行乘除运算时，如果后续指令需要用到乘除法部件，则需要暂停，等待乘除法部件运算完成。如果后续指令无需乘除法部件，则可以继续向后流水执行。

### （二）

举例说明并分析何时按字节访存相对于按字访存性能上更有优势。

C语言中一个字符占一个字节，在访问字符串时，字符串的长度不一定为4的倍数，此时若按字访问内存，会额外访问到不属于字符串的内容。若按字节访问内存，则不会额外访问到不属于字符串的内容。

### （三）

在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

跳转指令和计算指令，尤其是jal指令和计算指令。我将pc+8也作为可能写入寄存器的值向后流水，pc+8可以在E、M、W三级流水线被转发。

如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证**覆盖**了所有需要测试的情况；如果你是**完全随机**生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了**特殊的策略，**比如构造连续数据冒险序列，请你描述一下你使用的策略如何**结合了随机性**达到强测的效果。

我的构造策略是：假设发生冲突的新指令已经到达了要使用阶段，如addu已经到达了E级，便在addu前四条指令的范围内构造冲突指令，如想测试M级向ALU的转发，则在其前紧跟一条计算指令，如想测试W级向ALU的转发，则在其前第二条指令处加计算指令或lw指令。通过依次枚举，可以覆盖所有需要测试的情况。

### （四）

为了对抗复杂性你采取了哪些抽象和规范手段，这些手段在译码和处理数据冲突的时候有什么样的特点和帮助？

在译码时，我将指令进行了分类，分为了add类、addi类、sw类、lw类、beq类、j类。进行分类后，各类中的指令特点和行为类似，减少了控制信号的数量，在编写控制信号时也变得更加简单便捷，使实现思路更加简单和清晰。

在处理数据冲突时，也按照指令的分类给出T\_new与T\_use，使处理数据冲突的思路同P5大体一致，使实现变得更加简单。