



**计算机组成原理 课 程 设 计**

个人实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 | 王娟 |
| 组 长 | 宋尚儒 |
| 组 员 | 陈鸿韬、赵旭 |
| 组长联系方式 | 1120180717@bit.edu.cn |

二O二一年 九月

目 录

[第一章 项目简述 1](#_Toc84763288)

[第二章 设计目的 1](#_Toc84763289)

[第三章 设计环境 1](#_Toc84763290)

[第四章 设计原理及内容 1](#_Toc84763291)

[4.1 数据通路 2](#_Toc84763292)

[4.2 控制逻辑 2](#_Toc84763293)

[第五章 设计与实现 4](#_Toc84763294)

[第六章 测试 5](#_Toc84763295)

[第七章 问题及解决方法 7](#_Toc84763296)

[第八章 心得体会及总结 7](#_Toc84763297)

[第九章 参考文献有价值的资源推荐 7](#_Toc84763298)

# 项目简述

实现支持MIPS指令子集的单周期CPU，基础要求的指令外，实现了addi和addu（随机抽取）指令，给出了覆盖每一条指令的详细验证程序，结果正确。

# 设计目的

利用Verilog语言，设计并实现一个支持MIPS指令子集：Lui，Addiu，Add，Lw，Sw，Beq，j，以及一条随机抽取的指令的单周期CPU，给出详细的测试程序，完成实验报告以及提交物

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows10 |
| 编程语言 | Verilog HDL |
| EDA工具 | Vivado2019.2 |
| 汇编语言 | MIPS |
| 汇编程序编辑器 | mars4\_5 |

请标注版本号

# 设计原理及内容

本流水线CPU覆盖9条指令(见表5.1-1)，其中7条为基础要求指令，1条为随机抽取的指令，1条是自行实现的指令，下面将从数据通路和控制逻辑阐述设计思路。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | function | 功能 |
| 1 | ADDI | 001000 | X | rd=rs+im |
| 2 | ADD | 000000 | 100000 | rd=rs+rt |
| 3 | LUI | 001111 | X | rt=im\*65536 |
| 4 | ADDIU | 001001 | X | rd=rs+im（无符号数） |
| 5 | ADDU | 000000 | 100001 | rd=rs+rt (无符号数) |
| 6 | SW | 101011 | X | SW rt, offset(base) |
| 7 | LW | 100011 | X | LW rt, offset(base) |
| 8 | BEQ | 000100 | / | PC=(rs==rt)?PC+im<<2:PC |
| 9 | J | 000010 | X | PC={(PC+4)[31,28],addr,00} |
| 0 | NULL | X | X |  |

表5-1 9条指令，红色为随机抽取指令，绿色为自行实现指令

## 数据通路

图示, 示意图

描述已自动生成

图5.1 单周期CPU数据通路

其中蓝色表示大部分由CU模块控制的控制信号，因为连线比较复杂所以不直接表现连接关系，红色表示ALU计算是否为0信号，具体会在之后说明。

## 控制逻辑

**控制信号列表：**

**pc\_jmp**：pc执行jmp跳转，0-不执行，1-执行；对于JMP指令该值为1；

**pc\_beq**：pc执行beq跳转，0-执行，1-执行；根据之后的cu\_br和alu\_zero信号的与值判断是否执行beq跳转。

**reg\_we**：寄存器组写使能，0-不能写，1-能写；除SW、BEQ、J指令外均为不能写；

**alu\_op**：alu中执行的操作；

1. add；
2. sub；
3. lui；
4. null；

根据指令进行判断，其中add对应指令较多，sub仅有beq指令，lui仅有lui指令；

**d\_we**：数据存储器写使能，0-不能写，1-能写；指令为SW时为1；

**cu\_alu\_src**：ALU第一个操作数来源，0-寄存器，1-立即数；当指令为add、addu、beq、jmp时为0；

**cu\_wd\_dst**：写入的目的寄存器地址，0-rt对应寄存器，1-rd段对应寄存器；当指令为add和addu时为1；

**cu\_unsign\_ext**：符号扩展方式，0-带符号扩展，1-无符号扩展；当指令为addiu时为1；

**cu\_wd\_src**：向寄存器写的数据来源，0-alu运算结果，1-数据存储器读出结果；指令为LW时为1；

**cu\_br**：指令是否为beq，0-否，1-是；指令为beq时为1；

**alu\_zero**：ALU运算结果是否为0，0-否，1-是；

**数据信号列表：**

**pc\_val**：PC提供给指令存储器的当前指令地址。

**pc\_off**：PC执行JMP指令时的目标地址。

**pc\_tar**：PC执行BEQ指令时的偏移地址。

**reg\_ra1**：寄存器组选择寄存器信号1

**reg\_ra2**：寄存器组选择寄存器信号2

**reg\_wa**：写寄存器的选择寄存器信号

**reg\_wd**：写寄存器的数据

**reg\_rd1**：选择信号1对应的寄存器的值

**reg\_rd2**：选择信号2对应的寄存器的值

**alu\_in1**：ALU输入数据1

**alu\_in2**：ALU输入数据2

**alu\_out**：ALU计算结果

# 设计与实现

共设计并实现了5个模块，分别为

1. cpu：顶层模块，负责各个模块的连接，信号的选择；
2. regfile：通用寄存器堆，共32个寄存器；
3. control：控制核心模块，根据指令的opcode和func字段，确定指令的类型及功能，并产生相应的控制信号，如alu\_op、cu\_alu\_src等。其中部分指令和信号的关系采用掩码的方式实现；
4. alu：计算核心模块，进行算术运算，如加法、减法、拼接(LUI)、或运算等，并根据计算结果产生可能的控制信号，比如计算结果为0、溢出等；
5. pc：存储当前指令地址，根据两个控制信号分别进行递增4、JMP跳转、BEQ跳转等操作。

此外，对于部分指令所需的符号扩展，由于所需逻辑较为简单，在CPU模块中使用组合电路进行实现。

其接口定义如下

1. cpu(
2. input rstn,
3. input clk,
4. output [31:0] inst\_rom\_addr,
5. input [31:0] inst\_rom\_rdata,
6. output [31:0] data\_ram\_addr,
7. output [31:0] data\_ram\_wdata,
8. output data\_ram\_wen,
9. input [31:0] data\_ram\_rdata
10. );
11. pc(
12. input clk,
13. input rst,
14. input pc\_jmp,
15. input pc\_beq,
16. //beq
17. input [31:0] pc\_off,
18. //jmp
19. input [31:0] pc\_tar,
20. output [31:0] pc\_val
21. );
22. regfile(
23. input wire clk,
24. input wire rst,
25. input wire[4:0] reg\_ra1,
26. input wire[4:0] reg\_ra2,
27. input wire reg\_we,
28. input wire[4:0] reg\_wa,
29. input wire[31:0] reg\_wd,
30. output wire[31:0] reg\_rd1,
31. output wire[31:0] reg\_rd2
32. );
33. alu(
34. input wire clk,
35. input wire rst,
36. input wire[3:0] alu\_op,
37. input wire[31:0] alu\_in1,
38. input wire[31:0] alu\_in2,
39. output reg [31:0] alu\_out,
40. output wire alu\_zero
41. );
42. control(
43. input [5:0] opcode,
44. input [5:0] func,
46. output [3:0] cu\_cA,
47. output cu\_jmp,
48. output cu\_br,
49. output d\_we,
50. output reg\_we,
51. output cu\_alu\_src,
52. output cu\_wd\_dst,
53. output cu\_unsign\_ext,
54. output cu\_wd\_src
55. );

实现的逻辑具体实现细节可参考源码验证。

# 测试

编写的汇编程序使用了MIPS汇编设计的测试代码，具体的实现和验证方法均已在MIPS汇编设计报告中说明，此处仅展示模拟验证与CPU仿真测试的对比结果。

首先在MARS中进行验证，需要注意的是，由于数据地址空间起始地址不同，所以对于sw和lw涉及的地址在MARS验证和仿真验证中被设置为不同的值。

1. 如下图所示，在执行完成前五条指令后，$t1-$t5各寄存器对应的指令计算结果正确

图形用户界面, 应用程序

描述已自动生成

1. 如下图所示，$t4和$t6寄存器值相同

图形用户界面, 应用程序

描述已自动生成

1. 如下图所示，$t1值未改变，说明跳过了(addi $t1, $t1, 100)

图形用户界面, 应用程序

描述已自动生成

1. 如下图所示，循环执行两条指令，$t2递增1，$t1不变，与预期相符

图形用户界面, 应用程序

描述已自动生成

在转换为机器码coe文件并写入单周期CPU的IP核后，进行仿真测试，由于指令仅修改$t1-$t6以及$pc寄存器，所以仅需要观察这些寄存器的值即可，对应的信号为regfile的regs[9]-regs[14]和pc的pc\_val，仿真结果如下图所示：

图形用户界面

描述已自动生成

可以看到，pc寄存器结果正确反映了顺序执行、beq跳转、jmp跳转的执行结果；在完成前5个指令（开始执行后的五个周期）后，$t1-$t5寄存器对应值和修改时间也与MARS模拟相同；对比$t4和$t6寄存器值最终相同，说明sw和lw指令执行正确；$t2寄存器最后陷入循环自增，且$t1没有变化，说明jmp指令正确。

综上所述，设计的测试程序完整验证了单周期CPU的MIPS指令子集，测试结果正确，单周期CPU设计无误。

# 问题及解决方法

1. 问题：部分指令的立即数需要进行符号扩展，部分指令需要非符号扩展

解决：在CPU模块设置专用的组合电路与信号进行不同的符号扩展的处理，用以解决特定指令问题。

1. 问题：部分信号涉及多个指令，使用选择语句进行描述冗长

解决：采用掩码的形式描述某信号是否与某指令有关，虽然降低了可读性，但精简了控制模块的结构

# 心得体会及总结

虽然单周期CPU结构相对简单，但对于初学者来说，手动实现还是相当不易，在助教的ppt和示例代码的基础上设计实现了单周期CPU，并且设计了汇编程序进行测试，这让我对CPU的结构有了更清楚的认知，并学习了vivado等设计工具的使用方式。

# 参考文献有价值的资源推荐

无