



**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 |
| 姓 名 | 张驰 |
| 联系方式 | 18810575675 |

二O二二年 九月

目 录

# 项目简述

本项目实现了支持MIPS部分指令子集的单周期CPU。本项目实现的MIPS指令子集包含：Lui，Addiu，Add，Lw，Sw，Beq，J，以及一条随机抽取的指令（Srav指令）。

# 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（SRAV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

## CPU整体架构

本流水线CPU覆盖8条指令(见表4.1-1)，下面将从数据通路和控制逻辑阐述设计思路。

Lui，Addiu，Add，Lw，Sw，Beq，j，Srav

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 指令 | opcode | function | 功能 |
| 1 | LUI | 001111 | / | rt=im\*65536 |
| 2 | ADDIU | 001001 | / | rd=rs+im（无符号数） |
| 3 | ADD | 000000 | 100000 | rd=rs+rt |
| 4 | LW | 100011 | / | LW rt, offset(base) |
| 5 | SW | 101011 | X | SW rt, offset(base) |
| 6 | BEQ | 000100 | / | PC=(rs==rt)?PC+im<<2:PC |
| 7 | J | 000010 | X | PC={(PC+4)[31,28],addr,00} |
| 8 | SRAV | 000000 | 000111 | rd=rt>>rs（符号位保留） |

表4.1-1 本人设计的CPU覆盖的8条指令

### 数据通路

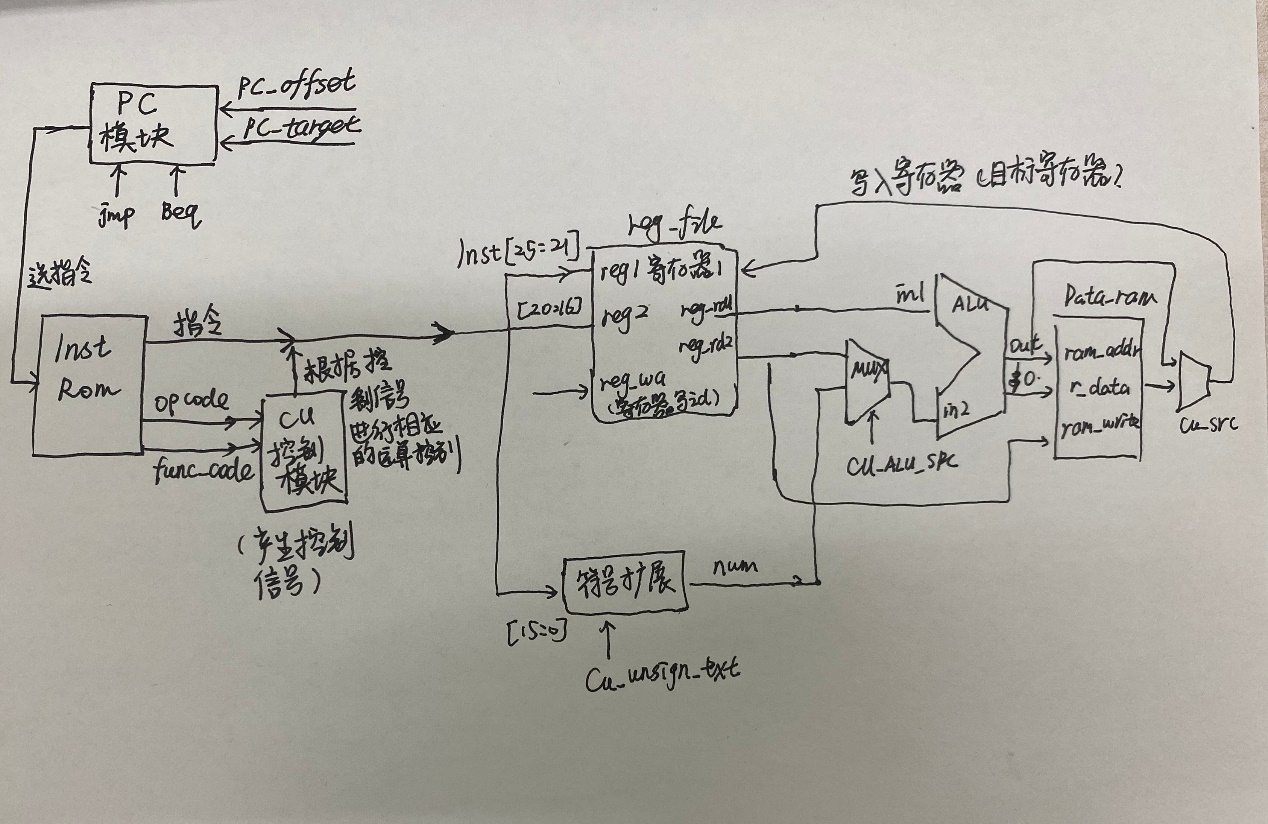


图4.1.1-1 单周期CPU数据通路

### 控制逻辑

分析实现控制逻辑采用的方式：

CPU模块下一共包含4个模块：

1. PC模块：PC模块是存储当前所指的指令地址模块，通过PC模块控制当前指令地址每次自增4。同时该模块的信号需要识别是否出现了J，Beq指令，从而实现转移指令，将PC模块的当前指令地址转移。

其中包含的控制信号如下：

Pc\_jmp : 含义为是否执行J指令跳转

Pc\_beq : 含义为是否执行Beq指令跳转。同时，该指令需要判断两个寄存器的内容是否相等，需要得到控制模块的信号，与其进行与操作进行判断是否跳转即可。

Pc\_tar : 含义为J指令跳转的目标地址。

Pc\_off : 含义为Beq指令跳转的偏移地址。

1. ALU模块：ALU是计算的核心模块，进行加减法、拼接、算术移位等运算，同时影响一些控制信号和寄存器。

Alu\_op：传递给alu，使alu执行的操作（add,sub,lui,srav,null）各类操作，add对应多数的指令，sub用于执行beq指令，lui用于执行lui指令，srav用于执行srav指令

Alu\_in1/2：输入数据

Alu\_out：计算结果

1. Regfile模块：主要用于控制32个通用寄存器，在本代码中主要使用9号-14号寄存器。

Reg\_write\_enable：寄存器组是否可以写入的使能信号；当指令为SW,BEQ,J指令时，寄存器组可以进行写入。

Reg\_read\_data1/2：选择信号1、2对应的寄存器的值，读取的寄存器

Reg\_write\_data：写寄存器的数据

Reg\_write\_addr：写寄存器的信号

1. Cu模块：控制模块，根据opcode和funccode，确定机器码对应的指令和功能，产生相应的控制信号。

Cu\_wd\_dst：写入目的寄存器的地址，0为rt对应寄存器，1为rd对应寄存器

Cu\_wd\_src：写入目的寄存器的数据来源，0为alu中的运算结果，1为数据存储器读出的结果。

Cu\_beq：是否为beq指令

Cu\_ext：符号扩展方式，0为带符号扩展，1为无符号扩展，addiu采用无符号扩展，srav采用带符号扩展。

# 设计与实现

CPU模块是顶层模块，包含四个子模块：PC模块、ALU模块、Regfile模块、Cu模块。CPU模块用于各个模块之间的信号传递，模块之间的连接。CPU统领各个模块的配合，信号的传递，从而实现各个模块之间的配合。

CPU模块下一共包含4个模块：

1. PC模块：PC模块是存储当前所指的指令地址模块，通过PC模块控制当前指令地址每次自增4。同时该模块的信号需要识别是否出现了J，Beq指令，从而实现转移指令，将PC模块的当前指令地址转移。
2. ALU模块：ALU是计算的核心模块，进行加减法、拼接、算术移位等运算，同时影响一些控制信号和寄存器。

ALU模块包含一个子模块，用于执行SRAV指令。由于算术右移模块，需要采用时序逻辑，所以这里专门为了SRAV指令构建一个模块，用于向右算术移位指令。

1. Regfile模块：主要用于控制32个通用寄存器，在本代码中主要使用9号-14号寄存器。
2. Cu模块：控制模块，根据opcode和funccode，确定机器码对应的指令和功能，产生相应的控制信号。

CPU下各个模块中的配合代码如下所示：

1. module cpu(
2. input       rst\_cpu,
3. input       clk,
4. //指令接口
5. output [31:0] inst\_rom\_addr,
6. input  [31:0] inst\_rom\_rdata,
7. //数据接口
8. output [31:0] data\_ram\_addr,
9. output [31:0] data\_ram\_wdata,
10. output        data\_ram\_wen,
11. input  [31:0] data\_ram\_rdata
12. );
13. pc pc(
14. .clk(clk),
15. .rst(rst\_cpu),
16. .pc\_jmp(pc\_jmp\_instruction),
17. .pc\_beq(pc\_beq\_instruction),
18. .pc\_off(pc\_offset),
19. .pc\_tar(pc\_tar),
20. .pc\_val(pc\_val\_addr)
21. );
22. regfile regfile(
23. .clk(clk),
24. .rst(rst\_cpu),
25. .reg\_we(reg\_write\_enable),
26. .reg\_ra1(reg\_ra1),
27. .reg\_ra2(reg\_ra2),
28. .reg\_wa(reg\_write\_addr),
29. .reg\_wd(reg\_write\_data),
30. .reg\_rd1(reg\_read\_data1),
31. .reg\_rd2(reg\_read\_data2)
32. );
34. //alu
35. alu alu(
36. .clk(clk),
37. .rst(rst\_cpu),
38. .alu\_op(alu\_op),
39. .alu\_in1(alu\_in1),
40. .alu\_in2(alu\_in2),
41. .alu\_out(alu\_out),
42. .alu\_zero(alu\_zero)
43. );
45. //cu
46. control cu(
47. .opcode(opcode),
48. .func\_code(func\_code),
49. .cu\_cA(alu\_op),
50. .cu\_jmp(pc\_jmp),
51. .cu\_br(cu\_br),
52. .d\_we(d\_we),
53. .reg\_we(reg\_we),
54. .cu\_alu\_src(cu\_alu\_src),
55. .cu\_wd\_dst(cu\_wd\_dst),
56. .cu\_unsign\_ext(cu\_unsign\_ext),
57. .cu\_wd\_src(cu\_wd\_src)
58. );
59. endmodule

模块内的具体代码不再展示。

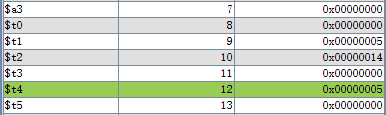
# 测试

测试该单周期CPU的汇编程序使用MIPS汇编设计的测试代码，使用的MIPS汇编测试代码包含了所有的测试指令，MIPS汇编代码如下所示：

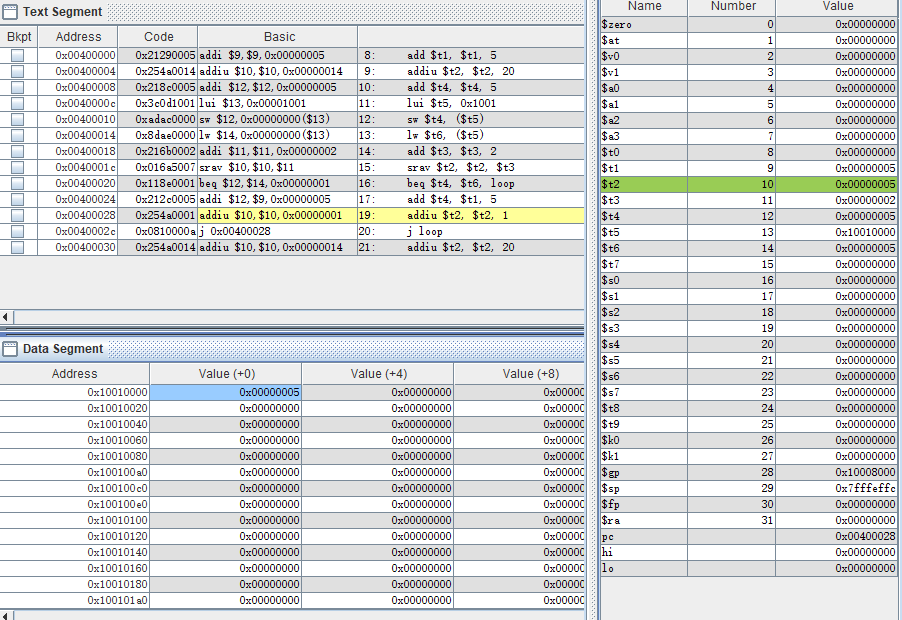
1. .org 0x0
2. .set noat
3. .set noreorder
4. .set nomacro
5. .global \_start
7. \_start:
8. add $t1, $t1, 5
9. addiu $t2, $t2, 20
10. add $t4, $t4, 5
11. lui $t5, 0xbfa1
12. sw $t4, ($t5)
13. lw $t6, ($t5)
14. add $t3, $t3, 2
15. srav $t2, $t2, $t3
16. beq $t4, $t6, loop
17. add $t4, $t1, 5
18. loop:
19. addiu $t2, $t2, 1
20. j loop
21. addiu $t2, $t2, 20

首先，在Mars4.5中对该程序进行验证，验证结果通过寄存器及跳转来展示。通过该汇编程序后，各个寄存器的变化情况如下所示：

1. 运行第一条指令后，寄存器$t1的值变为5，并会一直维持到最后
2. 运行第二条和第三条指令：$t2的值变为0x00000014，$t4的值变为5。



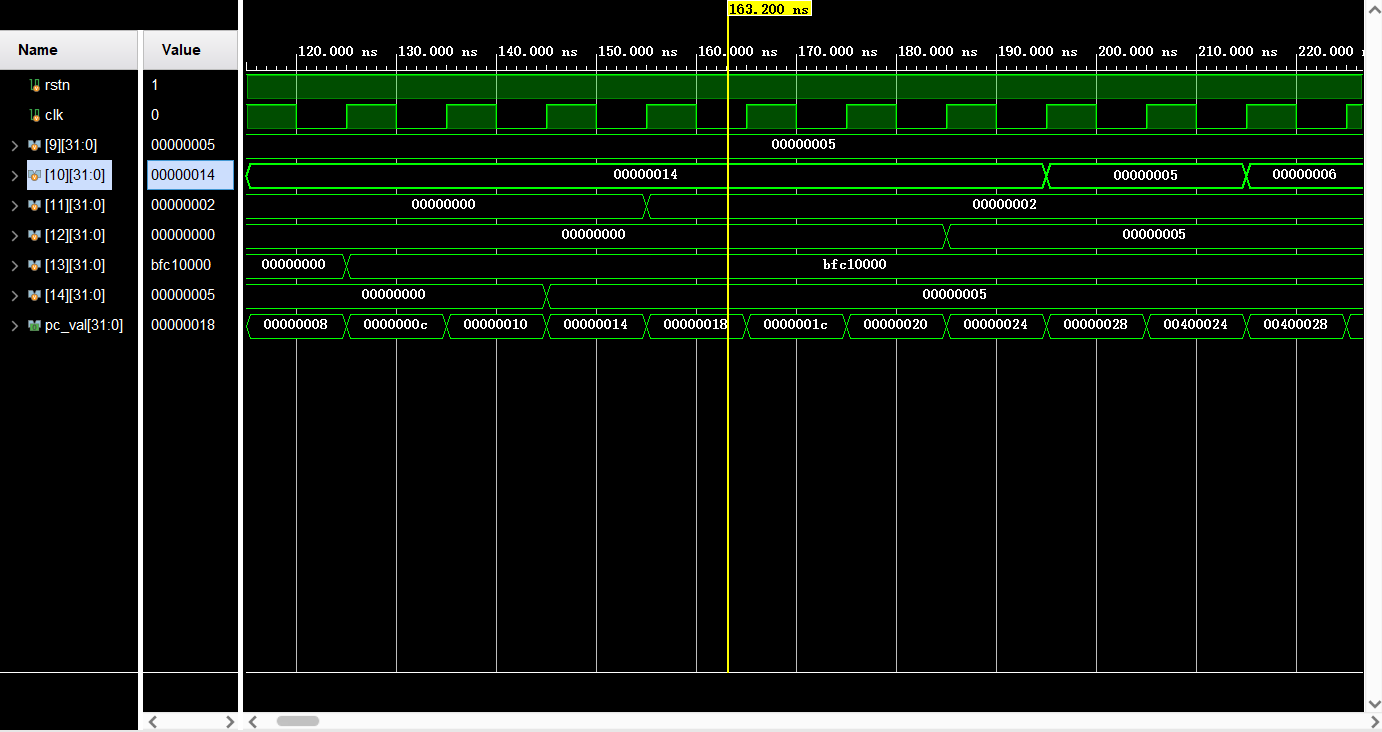
1. Sw和Lw指令，分别使寄存器$t5的值变为0x10010000，内存空间0xbfa10000的值为0x00000005。
2. 后面的指令配合SRAV指令，使寄存器$t2向右进行算术移位2位，变为0x00000005。



1. 由于寄存器$t4和$t6值相同，Beq指令运行后会直接跳转至Loop，PC会发生相应的变化，之后进行循环，每次循环寄存器$t2自增1。

将该汇编程序转换为机器码，写入单周期CPU的IP核中，运行仿真测试，观察pc寄存器以及寄存器9号-寄存器14号的值。

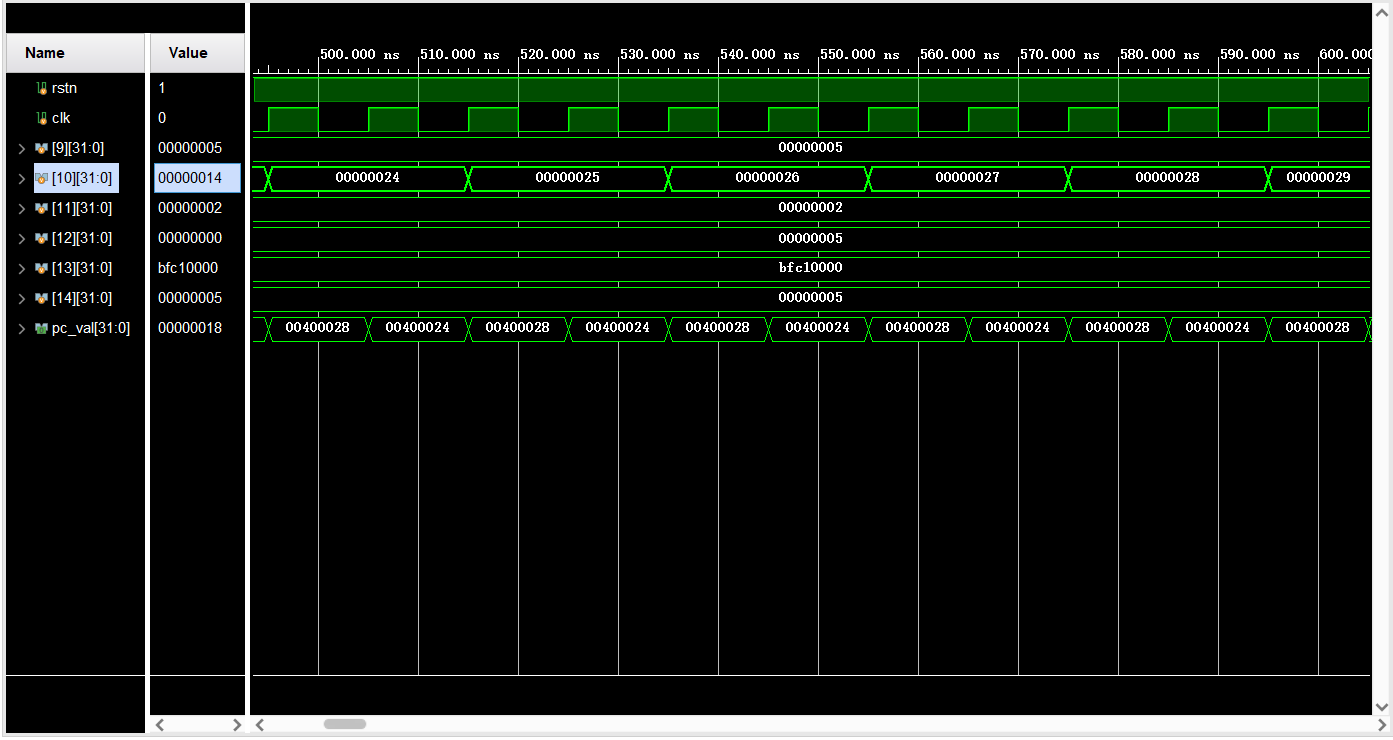
仿真结果如下图所示：



可以看到，pc值在不发生跳转的情况下，每次增长4；

同时，在10号寄存器（$t2）中，因为发生了一次移位，所以值由0x00000014，变为 0x00000005，随后自增。在9号寄存器（$t1）中，值从头到尾固定为0x00000005。在经过sw和lw指令后，$t5的值为0xbfc10000，说明语句成功。且$t4寄存器和$t6寄存器的内容相同。

所有语句结束后，开始进行循环，仿真截图如下所示：



寄存器$t2随着时间的变化自增，pc寄存器的值在0x00400028和0x00400024两个数之间变动，其余寄存器的值稳定。

仿真结果正确，验证无误。

# 问题及解决方法

1. 问题及解决：在进行算术移位时，由于算术移位的逻辑较为复杂，所以采用专门的模块：shifter\_right来解决该问题。同时，由于该指令为算术运算指令，所以在进行符号扩展时，需要使用带符号扩展，通过CU模块进行控制。在移位过程中，采用$signed(d) >>> sa 的形式对其进行移位。
2. 问题及解决：CU的控制信号涉及到多个指令的传输，通过掩码的形式描述该控制信号是否与当前所运行的指令相关，从而简化了控制结构。

# 心得体会及总结

单周期CPU的整体逻辑比较清晰，结构较为简单。但是本人在手动实现的过程中，依然遇到了很多的bug和问题。配合助教的ppt，示例代码以及网上的资源，本人最终完成了MIPS指令子集的单周期CPU设计，同时设计了相应的汇编程序进行测试。

通过本次个人实验完成CPU的过程，我对计算机组成原理、计算机体系结构的相关知识有了进一步的了解，同时对Verilog HDL的语法和Vivado的相关操作进一步熟悉，为组队实验的完成做好了铺垫。

# 参考文献有价值的资源推荐

无