

SHANGHAI JIAOTONG UNIVERSITY



计算机系统结构实验报告 – Lab5

仿真测试 & 下载验证

姓名：刘欣鹏

学号：516030910259

完成时间：2018/4/4

目录

[1. 概述 4](#_Toc20457)

[1.1 实验名称 4](#_Toc19106)

[1.2 实验目的 4](#_Toc25354)

[1.3实验范围 4](#_Toc7756)

[2. 实验描述 4](#_Toc17543)

[2.1 顶层模块Top 4](#_Toc22918)

[2.1.1模块描述 4](#_Toc5226)

[2.1.2 Top模块代码 5](#_Toc840)

[2.2 Program Counter模块 7](#_Toc28795)

[2.2.1 模块描述 7](#_Toc29225)

[2.2.2 PC模块代码 7](#_Toc27965)

[2.3 Inst\_MEMORY模块 8](#_Toc10202)

[3.仿真 8](#_Toc12265)

[3.1 仿真代码 9](#_Toc32182)

[3.2 仿真波形 9](#_Toc31061)

[4. 下载验证 9](#_Toc21143)

[4.1 实验描述 9](#_Toc12510)

[4.2 模块改进 9](#_Toc11411)

[4.2.1 分频器模块 10](#_Toc14284)

[4.2.2 Top模块改进 10](#_Toc5447)

[4.3 管脚定义 11](#_Toc20694)

[4.4 实验结果 11](#_Toc28123)

[5. 实验心得 11](#_Toc1187)

# 1. 概述

## 1.1 实验名称

简单的类MIPS单周期处理器实现——整体调试

## 1.2 实验目的

完成单周期的类MIPS处理器

## 1.3实验范围

本次实验将覆盖以下范围:

(1) ISE的使用

(2) Xilinx Spartan 3E实验板的使用

(3)使用VerilogHDL进行逻辑设计

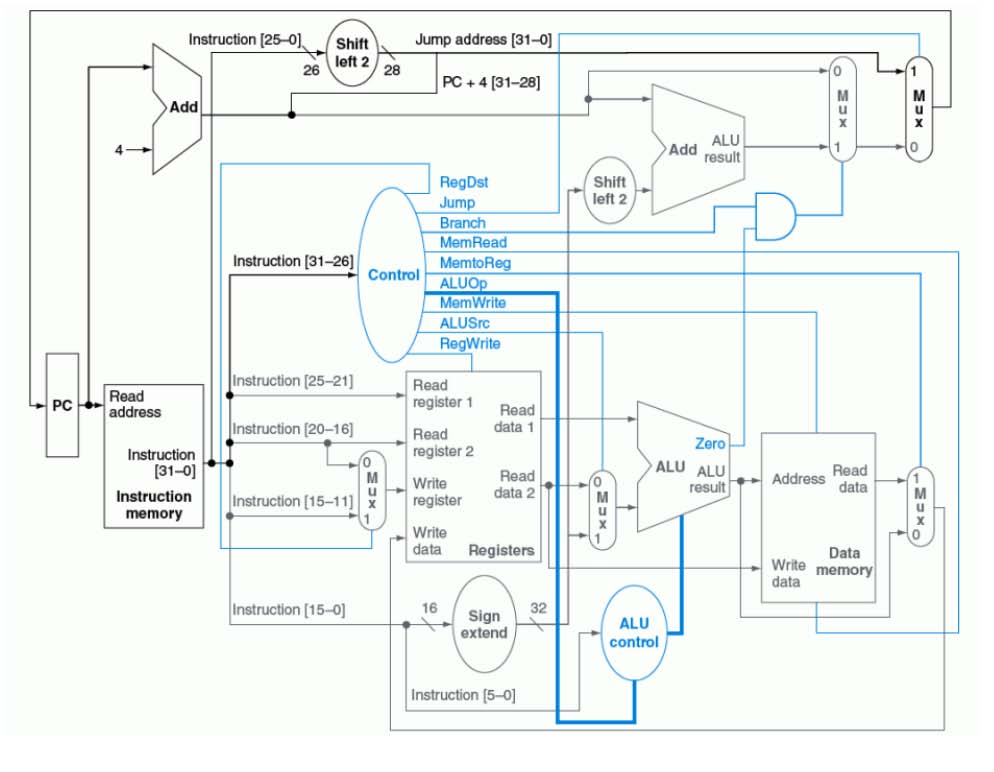
(4)仿真测试、下载验证

# 2. 实验描述

## 2.1 顶层模块Top

### 2.1.1模块描述

顶层模块将类MIPS单周期处理器的各个组成部分联结起来，形成完整的处理器。本次实验中，top模块内直接实现了multiplexer与adder模块的功能。



MIPS单周期处理器原理图

### 2.1.2 Top模块代码

**module top(**

**input reset,**

**input clock,**

**output wire [31:0] R0,**

**output wire [31:0] R1,**

**output wire [31:0] R2,**

**output wire [31:0] R3,**

**output wire [31:0] R4,**

**output wire [31:0] R5,**

**output wire [31:0] R6,**

**output wire [31:0] R7,**

**output wire [31:0] R8,**

**output wire [31:0] M0,**

**output wire [31:0] M1,**

**output wire [31:0] M2,**

**output wire [31:0] M3,**

**output wire [31:0] M4,**

**output wire [31:0] M5,**

**output wire [31:0] M6,**

**output wire [31:0] M7,**

**output wire [31:0] M8,**

**output reg [31:0] PC,**

**output wire [31:0] INST**

**);**

**wire [3:0] alu\_CTR;**

**wire ZERO, reg\_DST, alu\_SRC, memTo\_REG, reg\_WRITE, mem\_READ, mem\_WRITE, BRANCH, JUMP;**

**wire [1:0] alu\_OP;**

**wire [31:0] DATA, read\_DATA1, read\_DATA2, read\_DATA, alu\_RES, regWrite\_DATA, INPUT2, EXTENDED;**

**wire [4:0] write\_REG;**

**assign write\_REG=reg\_DST?INST[15:11]:INST[20:16];**

**assign regWrite\_DATA=memTo\_REG?read\_DATA:alu\_RES;**

**assign INPUT2=alu\_SRC?EXTENDED:read\_DATA2;**

**programCounter mainPC(**

**.reset(reset),**

**.clock(clock),**

**.branch(BRANCH),**

**.jump(JUMP),**

**.INST(INST),**

**.zero(ZERO),**

**.PC(PC)**

**);**

**signext mainSignext(**

**.inst(INST[15:0]),**

**.extend(EXTENDED)**

**);**

**inst\_memory mainRom(**

**.pc(PC),**

**.inst(INST)**

**);**

**ctr mainCtr(**

**.opCode(INST[31:26]),**

**.regDst(reg\_DST),**

**.jump(JUMP),**

**.branch(BRANCH),**

**.memRead(mem\_READ),**

**.memToReg(memTo\_REG),**

**.aluOp(alu\_OP),**

**.memWrite(mem\_WRITE),**

**.aluSrc(alu\_SRC),**

**.regWrite(reg\_WRITE)**

**);**

**AluCtr mainAluCtr(**

**.aluOp(alu\_OP),**

**.funct(INST[5:0]),**

**.aluCtr(alu\_CTR)**

**);**

**alu mainAlu(**

**.input1(read\_DATA1),**

**.input2(INPUT2),**

**.aluCtr(alu\_CTR),**

**.zero(ZERO),**

**.aluRes(alu\_RES)**

**);**

**data\_memory mainDataMemory(**

**.clock\_in(clock),**

**.address(alu\_RES),**

**.writeData(read\_DATA2),**

**.memRead(mem\_READ),**

**.memWrite(mem\_WRITE),**

**.readData(read\_DATA),**

**.mem0(M0),**

**.mem1(M1),**

**.mem2(M2),**

**.mem3(M3),**

**.mem4(M4),**

**.mem5(M5),**

**.mem6(M6),**

**.mem7(M7),**

**.mem8(M8)**

**);**

**register mainRegister(**

**.clock\_in(clock),**

**.readReg1(INST[25:21]),**

**.readReg2(INST[20:16]),**

**.writeReg(write\_REG),**

**.writeData(regWrite\_DATA),**

**.regWrite(reg\_WRITE),**

**.reset(reset),**

**.readData1(read\_DATA1),**

**.readData2(read\_DATA2),**

**.reg0(R0),**

**.reg1(R1),**

**.reg2(R2),**

**.reg3(R3),**

**.reg4(R4),**

**.reg5(R5),**

**.reg6(R6),**

**.reg7(R7),**

**.reg8(R8)**

**);**

**endmodule**

## 2.2 Program Counter模块

### 2.2.1 模块描述

程序计数器PC是这个简单CPU能够正常运行的关键所在，PC是一个32位reg类型的变量，在时钟上升沿（下降沿已经被用作寄存器的写了）做PC <= PC + 4，或执行分支/跳跃指令。

### 2.2.2 PC模块代码

**module programCounter(**

**input reset,**

**input clock,**

**input branch,**

**input jump,**

**input [31:0] INST,**

**input zero,**

**output reg [31:0] PC**

**);**

**initial begin**

**PC=0;**

**end**

**always @ (posedge clock)**

**begin**

**if (reset) PC<=0;**

**else if (zero && branch) PC<=PC+4+(INST[15:0]<<2);**

**else if (jump) PC<=(INST[25:0]<<2);**

**else PC<=PC+4;**

**end**

**Endmodule**

## 2.3 Inst\_MEMORY模块

Instruction Memory模块与Data\_Memory模块类似，也同样需要初始化，但它被用于存储MIPS指令，而且是只读的。代码如下：

**module inst\_memory(**

**input [31:0] pc,**

**output reg [31:0] inst**

**);**

**reg [31:0] rom[15:0];**

**//reg [31:0] inst;**

**initial begin**

**rom[0]=32'b00001000000000000000000000000001;//j 1**

**rom[1]=32'b10001100000000010000000000000001;//lw $1,1($0)**

**rom[2]=32'b10001100000000100000000000000011;//lw $2,3($0)**

**rom[3]=32'b00000000001000100001100000100000;//add $3,$1,$2**

**rom[4]=32'b00000000010000010010000000100010;//sub $4,$2,$1**

**rom[5]=32'b00000000011001000010100000101010;//slt $5,$3,$4**

**rom[6]=32'b00000000001000100011000000100100;//and $6,$1,$2**

**rom[7]=32'b00000000001000100011100000100101;//or $7,$1,$2**

**rom[8]=32'b10101100000000110000000000000001;//sw $3,1($0)**

**rom[9]=32'b10101100000001000000000000000010;//sw $4,2($0)**

**rom[10]=32'b10001100000010000000000000000001;//lw $8,1($0)**

**rom[11]=32'b10001100000000010000000000000010;//lw $1,2($0)**

**rom[12]=32'b00010000011010000000000000000001;//beq $8,$2,1**

**rom[13]=32'b00001000000000000000000000000000;//j 0**

**rom[14]=32'b00001000000000000000000000000001;//j 1**

**end**

**always @(pc) inst=rom[pc>>2];**

**endmodule**

# 3.仿真

## 3.1 仿真代码

仿真代码主要部分如下，完整代码见test\_ top.v。

**always #100 clock=~clock;**

**initial begin**

**clock = 0;**

**reset = 1;**

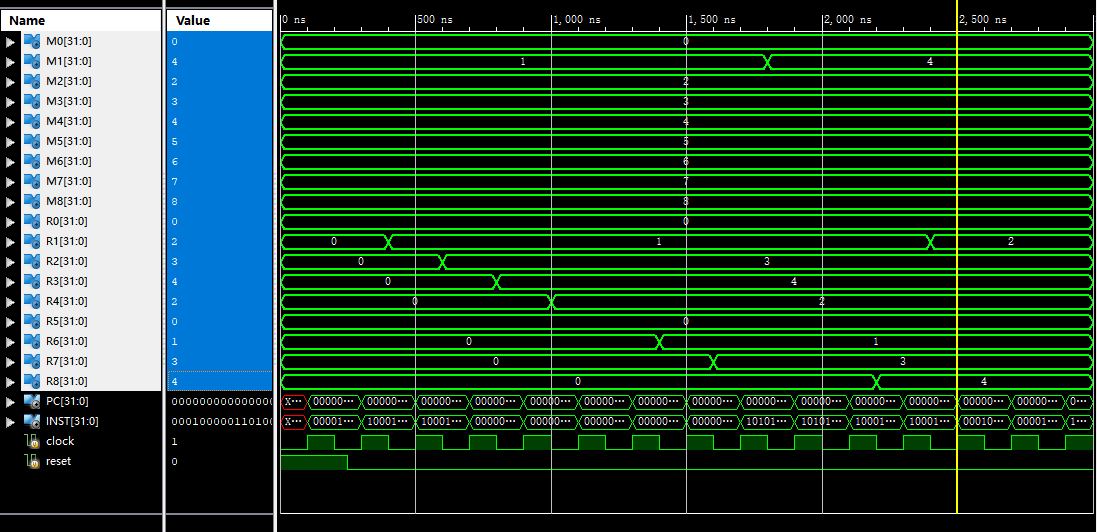
**#250;**

**reset = 0;**

**#3000;**

**end**

## 3.2 仿真波形



仿真波形说明Top模块能够正确更新Program Counter，读取Instruction Memory，并根据指令进行相应ALU计算、寄存器与内存读写等操作。Top模块仿真实验成功。

# 4. 下载验证

## 4.1 实验描述

为在Spartan 3E实验板上模拟这一类MIPS处理器的运行，我们需要将实验板的内部时钟进行分频以便于观察，同时将板子上4个开关分别用作reset和一个用作寄存器地址的3位二进制数。LED灯位从高到低分别表示：时钟信号，reset信号，所显示的寄存器地址，寄存器低三位的值。由于时钟信号下降沿被用来写寄存器，所以读取在上升沿进行。

## 4.2 模块改进

### 4.2.1 分频器模块

将时钟频率减缓226倍。

**module time\_divider(**

**input clock\_in,**

**output reg clock\_out**

**);**

**reg [25:0] buffer;**

**initial begin buffer=0; end**

**always@(posedge clock\_in)**

**begin**

**buffer<=buffer+1;**

**clock\_out<=buffer[25];**

**end**

**endmodule**

### 4.2.2 Top模块改进

新增sw作为地址输入、LED作为寄存器输出。clockin为实验板晶振时钟，分频后得到clock才可用于观察。此外，还需要添加在时钟信号变化沿更新LED状态的代码。详见top.v。

**initial begin**

**PC=0;LED=0;**

**end**

**time\_divider mainTime(**

**.clock\_in(clockin),**

**.clock\_out(clock)**

**);**

**always @ (posedge clock)**

**begin**

**if (reset) PC<=0;**

**else if (ZERO && BRANCH) PC<=PC+4+(INST[15:0]<<2);**

**else if (JUMP) PC<=(INST[25:0]<<2);**

**else PC<=PC+4;**

**end**

**always @ (clock)**

**begin**

**case(sw[2:0])**

**3'b000: LED[2:0]<=R0[2:0];**

**3'b001: LED[2:0]<=R1[2:0];**

**3'b010: LED[2:0]<=R2[2:0];**

**3'b011: LED[2:0]<=R3[2:0];**

**3'b100: LED[2:0]<=R4[2:0];**

**3'b101: LED[2:0]<=R5[2:0];**

**3'b110: LED[2:0]<=R6[2:0];**

**3'b111: LED[3:0]<=R7[2:0];**

**endcase**

**LED[6]<=reset;**

**LED[5:3]<=sw[2:0];**

**end**

**always @ (posedge clock) begin LED[7]=~LED[7];end**

## 4.3 管脚定义

NET "LED[0]" LOC = F12;

NET "LED[1]" LOC = E12;

NET "LED[2]" LOC = E11;

NET "LED[3]" LOC = F11;

NET "LED[4]" LOC = C11;

NET "LED[5]" LOC = D11;

NET "LED[6]" LOC = E9;

NET "LED[7]" LOC = F9;

NET "sw[0]" LOC = L13 | IOSTANDARD = LVTTL | PULLUP;

NET "sw[1]" LOC = L14 | IOSTANDARD = LVTTL | PULLUP;

NET "sw[2]" LOC = H18 | IOSTANDARD = LVTTL | PULLUP;

NET "reset" LOC = N17 | IOSTANDARD = LVTTL | PULLUP;

NET "LED[0]" SLEW = SLOW;

NET "LED[0]" IOSTANDARD = LVTTL;

NET "LED[0]" DRIVE = 8;

NET "LED[1]" SLEW = SLOW;

NET "LED[1]" IOSTANDARD = LVTTL;

NET "LED[1]" DRIVE = 8;

NET "LED[2]" SLEW = SLOW;

NET "LED[2]" IOSTANDARD = LVTTL;

NET "LED[2]" DRIVE = 8;

NET "LED[3]" SLEW = SLOW;

NET "LED[3]" IOSTANDARD = LVTTL;

NET "LED[3]" DRIVE = 8;

NET "LED[4]" SLEW = SLOW;

NET "LED[4]" IOSTANDARD = LVTTL;

NET "LED[4]" DRIVE = 8;

NET "LED[5]" SLEW = SLOW;

NET "LED[5]" IOSTANDARD = LVTTL;

NET "LED[5]" DRIVE = 8;

NET "LED[6]" SLEW = SLOW;

NET "LED[6]" IOSTANDARD = LVTTL;

NET "LED[6]" DRIVE = 8;

NET "LED[7]" SLEW = SLOW;

NET "LED[7]" IOSTANDARD = LVTTL;

NET "LED[7]" DRIVE = 8;

NET "clockin" IOSTANDARD = LVCMOS33;

NET "clockin" LOC = C9;

## 4.4 实验结果

reset置为1后，拨动三个开关在8个寄存器之间选择，可观察到reg[1]至reg[7]依次产生数值，且经比对与仿真结果完全一致，而reg[0]始终为0。reset重新置为0后，寄存器全部清零，再次置为1可重新观察数值变化。由此，类MIPS单周期处理器上板模拟实验成功实现了。

# 5. 实验心得

本次实验与之前的实验相比复杂得多，总的来说，实验的难点有两处：对MIPS单周期处理器原理的掌握以及对Verilog语言的熟练使用。

尽管计算机系统结构的理论课程还没有介绍完MIPS，但参考先前的CTR、ALU、寄存器、内存等模块的接口，但实验指导书为我们提供了完整的接线图，对实验的成功起到了很大的帮助。

由于MIPS单周期处理器整体结构比较复杂，线路繁杂，对各个模块、每条线的命名都需要遵循规范，抱持良好的代码风格，以保证正确性。

这次实验的成功除了个人的努力外，也得益于实验指导书的详尽介绍和老师的指导。在此感谢老师和同学们的帮助。