**《计算机组成原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | |  | | | **姓名**  **与学号** |  |
| **实验题目** | **实验三 简易单周期CPU实验** | | | | | |
| **实验时间** | **2020年** | | **实验地点** | **DS1410** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 🗹设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  （1）掌握不同类型指令在数据通路中的执行路径。  （2）掌握Vivado仿真方式。 | | | | | | |
| 二、实验项目内容  **1 实验内容**  **本实验包含以下模块：**  **(1) Datapath，其中主要包含 alu，PC，adder、mux2、signext（扩展）、sl2（移位）**  **(2) Controller(实验二已完成)，其中包含两部分，分别为maindec，aludec。**  **(3) 指令存储器instruction\_rom，数据存储器data\_ram（在Mems项目内，详见附带的项目文件）。使用Block Memory Generator IP构造。加载coe文件。**  **(4) mips将datapath与controller连接起来，而top文件将两个存储器与mips连接起来。连接示意图如下：**    **2 实验设计**  **2.1 数据通路**  **2.1.1 功能描述**  **数据通路包括寄存器堆、ALU、加法器、多路选择器、移位器与符号拓展模块。负责根据传入的信号及数据进行运算，同时可以将运算结果存入寄存器或输出到存储器中。**  **2.1.2 接口定义**   |  |  |  |  | | --- | --- | --- | --- | | 接口信号名 | 方向 | 宽度 | 含义 | | clk | input | 1-bit | 时钟信号 | | rst | input | 1-bit | 刷新信号 | | instr | input | 32-bit | 指令 | | ram\_read\_data | input | 32-bit | 数据存储器传入的数据 | | MemtoReg | output | 1-bit | 略 | | MemWrite | output | 1-bit | 略 | | ALUSrc | output | 1-bit | 略 | | RegDst | output | 1-bit | 略 | | RegWrite | output | 1-bit | 略 | | PCSrc | output | 1-bit | 略 | | Jump | output | 1-bit | 略 | | ALUControl | output | 3-bit | 略 | | | | | | | |
| 三、实验过程或算法（源程序）  *datapath代码如下：*  **module datapath(**  **input clk,**  **input rst,**  **input [31:0]instr,**  **input [31:0]ram\_read\_data,**  **input MemtoReg,**  **input MemWrite,**  **input ALUSrc,**  **input RegDst,**  **input RegWrite,**  **input PCSrc,**  **input Jump,**  **input [2:0]ALUControl,**  **output [31:0] ALUOut,**  **output [31:0] WriteData,**  **output [31:0] address,**  **output zero**  **//,output [31:0]test\_jump\_address,**  **//output [31:0]test\_plus\_4\_address,**  **//output [31:0]test\_branch\_address,**  **//output [31:0]test\_shifted\_address,**  **//output [31:0]test\_immea,**  **//output [31:0]test\_reg\_out\_1**  **);**    **wire [4:0]rs;**  **wire [4:0]rt;**  **wire [4:0]rd;**  **wire [15:0]immeadiate;**  **wire [25:0]jump\_immea;**  **assign rs=instr[25:21];**  **assign rt=instr[20:16];**  **assign rd=instr[15:11];**  **assign immeadiate=instr[15:0];**  **assign jump\_immea=instr[25:0];**  **wire [4:0]reg\_write\_destination;**  **wire [31:0]reg\_data\_1;**  **wire [31:0]reg\_data\_2;**  **mux2 #(5)regdst(rt,rd,RegDst,reg\_write\_destination);**  **wire [31:0]reg\_write\_data;**  **mux2 #(32) memtoreg (ALUOut , ram\_read\_data, MemtoReg, reg\_write\_data ) ;**  **regfile regfile(**  **.clk(clk),**  **.we3(RegWrite),**  **.ra1(rs),.ra2(rt),.wa3(reg\_write\_destination),**  **.wd3(reg\_write\_data),**  **.rd1(reg\_data\_1),.rd2(reg\_data\_2)**  **);**  **assign WriteData=reg\_data\_2;**  **wire [31:0]extented\_immea;**  **signext signext(immeadiate,extented\_immea);**  **wire [31:0] alu\_datain\_2;**  **mux2 #(32) alusrc(reg\_data\_2,extented\_immea,ALUSrc,alu\_datain\_2);**  **alu alu(reg\_data\_1,alu\_datain\_2,ALUControl,ALUOut,zero);**  **//assign test\_immea=extented\_immea;**  **//assign test\_reg\_out\_1=reg\_data\_1;**  **wire [31:0] shifted\_extented\_immea;**  **sl2 sl2\_0(extented\_immea,shifted\_extented\_immea);**  **wire [31:0] next\_address;**  **pc pc(**  **.next\_address(next\_address),**  **.clk(clk),**  **.rst(rst),**  **.address(address)**  **);**  **wire [31:0] plus\_4\_address;**  **adder plus\_4\_adder(address,32'h4,plus\_4\_address);**  **wire [31:0] branch\_address;**  **adder branch\_adder(shifted\_extented\_immea, plus\_4\_address,branch\_address);**  **wire [31:0] tmp\_address;**  **mux2 #(32)pcsrc(plus\_4\_address,branch\_address,PCSrc,tmp\_address);**  **wire [31:0] shifted\_jump\_immea;**  **sl2 sl2\_1({6'h0,jump\_immea},shifted\_jump\_immea);**  **wire [31:0] jump\_address;**  **assign jump\_address={plus\_4\_address[31:28],shifted\_jump\_immea[27:0]};**  **//assign test\_jump\_address=jump\_address;**  **mux2 #(32)jump(tmp\_address,jump\_address,Jump,next\_address);**  **//assign test\_plus\_4\_address=plus\_4\_address;**  **//assign test\_branch\_address=branch\_address;**  **//assign test\_shifted\_address=shifted\_extented\_immea;**  **endmodule**  *top代码如下：*  **module top(**  **input wire clk,rst,**  **output wire [31:0] WriteData,ALUOut,**  **output wire MemWrite,**  **//output wire [31:0] test\_instr,**  **//output wire [31:0] test\_ram\_read\_data,**  **//output wire [31:0] test\_address,**  **/\*output test\_memtoreg,test\_alusrc,test\_regdst,test\_regwrite,test\_jump,test\_pcsrc,test\_zero,\*.**  **//output [2:0]test\_alucontrol,**  **//output [31:0]test\_jump\_address,**  **//output [31:0]test\_plus\_4\_address,**  **//output [31:0]test\_branch\_address,**  **//output [31:0]test\_shifted\_address,**  **//output [31:0]test\_immea,**  **//output [31:0]test\_reg\_out\_1**  **);**  **reg [2:0]count=2'h0;**  **reg clk0=0;**  **always @(posedge clk)**  **begin**  **if(count==7)**  **begin**  **count<=0;**  **clk0<=~clk0;**  **end**  **else**  **begin**  **count<=count+1;**  **end**  **end**  **wire[31:0] instr\_address,instr,ram\_read\_data;**  **mips mips(clk0,**  **rst,**  **instr,**  **ram\_read\_data,**  **MemWrite,**  **instr\_address,**  **ALUOut,**  **WriteData,**  **/\*test\_memtoreg,test\_alusrc,test\_regdst,test\_regwrite,test\_jump,test\_pcsrc,test\_zero,\*/**  **/\*test\_alucontrol,**  **test\_jump\_address,**  **test\_plus\_4\_address,**  **test\_branch\_address,**  **test\_shifted\_address,**  **test\_immea,**  **test\_reg\_out\_1\*/**  **);**  **instruction\_rom imem(.clka(clk),**  **.ena(1'b1),**  **.addra(instr\_address),**  **.douta(instr)**  **);**  **data\_ram dmem(.clka(clk),**  **.ena(MemWrite),**  **.addra(ALUOut),**  **.dina(WriteData),**  **.douta(ram\_read\_data)**  **);**  **//assign test\_instr=instr;**  **//assign test\_ram\_read\_data=ram\_read\_data;**  **//assign test\_address=instr\_address;**  **endmodule**  其中，灰色注释部分是为了仿真时方便观察而添加的中间变量。  红色部分的代码为时钟分频，由于本实验所使用的Vivado版本过于老旧，导致存储器的延迟达两个周期，为了保证取指和其余操作大致发生在同一周期内，所以必须时钟分频，让mips模块用较慢的时钟信号。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  上面两图是另写的仿真（参见sim\_for\_top.v文件）的波形图，test\_instr为指令（16进制），对比下图指令中的机器码（红色方框圈起来的指令为执行了的指令）可知，两个beq指令和j指令都跳转正确。  具体每条指令的控制信号这里不方便展示，如果想查看请参考附带项目文件中的波形图文件sim\_for\_top\_behav.wcfg。    下图为已给的testbench的运行结果，显示Simuliation succeeded，仿真成功。  C:\Users\windows10\Documents\Tencent Files\2937572301\Image\C2C\5N)3C~$N_ZCZ_`1_R~5(VHM.png | | | | | | |