**《计算机组成原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **计算机科学与技术5班、6班** | | | **姓名与学号** | **胡栋月20181778、彭燃20181719** |
| **实验题目** | **实验二 处理器译码实验** | | | | | |
| **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）掌握单周期CPU控制器的工作原理及其设计方法。  （2）掌握单周期CPU各个控制信号的作用和生成过程。  （3）掌握单周期CPU执行指令的过程。  （4）掌握取指、译码阶段数据通路执行过程 | | | | | | |
| 二、实验项目内容  1. PC。（pc.v文件）  D 触发器结构，用于储存 PC(一个周期)。有 3 个输入，分别为 clk, rst, next\_address, 分别连接时钟信号、复位信号和下一个指令位置的32位信号；有 2 个输出，为 address, ena, 分别传输当前存储的指令地址以及指令存储器使能端信号，分别连接指令存储器的 addra, ena 端 口；    2. 加法器。（pc\_adder文件）  用于计算下一条指令地址，有 1 个输入， 1 个输出，输入值为当前指令地址address，输出为address加上32’h4后的值，输出端口为next\_address；  3. Controller。（controller.v文件）  其中包含两部分：  (a). main\_decoder。（maindec.v文件）  负责判断指令类型，并生成相应的控制信号。 1 个输入，为指令 inst 的高 6 位 op。1个输出，为产生的9位的控制信号signals（main\_decoder产生的信号将在controller文件中拆分为MemtoReg、Branch、ALUOp等8个信号）；  (b). alu\_decoder。（aludec.v文件）  负责ALU模块控制信号的译码。实现2个输入， 1个输出，输入分别为 funct, ALUOp；输出为2位信号ALUControl；  (c). 除上述两个组件，controller文件调用两个decoder，对应实现opcode, funct输入信号， 并传入调用模块；对应实现控制信号及ALUControl，并连接至调用模块相应端口；  4. 指令存储器。（IP核instruction\_rom）  使用BlockMemoryGeneratorIP构造。IP在memories项目中（详见附带的项目文件）。加载了提供的coe文件；  5. 顶层文件top.v。  将PC、加法器、Controller和指令存储器连接，输入为clk、rst，输出为各个控制信号和ALUControl，以及控制七段数码管的seg和ans。同时利用display模块将指令显示到七段数码管上；  6. display.v与seg7.v文件  实现将指令输出显示到七段数码管中。 | | | | | | |
| 三、实验设计  1. 控制器(Controller)  1.1 功能描述  根据传入的opcode和funct字段，输出各个控制信号以及ALUControl信号。  1.2 接口定义   |  |  |  |  | | --- | --- | --- | --- | | 信号名 | 方向 | 位宽 | 功能描述 | | opcode | input | 6-bit | 传入指令的31到26位 | | funct | input | 6-bit | 传入指令的5到0位 | | MemReg | output | 1-bit | 略 | | MemWrite | output | 1-bit | 略 | | ALUSrc | output | 1-bit | 略 | | RegDst | output | 1-bit | 略 | | RegWrite | output | 1-bit | 略 | | Branch | output | 1-bit | 略 | | Jump | output | 1-bit | 略 | | ALUControl | output | 3-bit | 略 |   1.3 逻辑控制  真值表   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 指令 | opcode | RegWrite | RegDst | MemWrite | MemtoReg | | R型 | 000000 | 1 | 1 | 0 | 0 | | lw | 100011 | 1 | 0 | 0 | 1 | | sw | 101011 | 0 | X | 1 | X | | beq | 000100 | 0 | X | 0 | X | | addi | 001000 | 1 | 0 | 0 | 0 | | j | 000010 | 0 | X | 0 | X |   真值表（续）   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 指令 | opcode | ALUSrc | Branch | Jump | ALUOp | | R型 | 000000 | 0 | 0 | 0 | 10 | | lw | 100011 | 1 | 0 | 0 | 00 | | sw | 101011 | 1 | 0 | 0 | 00 | | beq | 000100 | 0 | 1 | 0 | 01 | | addi | 001000 | 1 | 0 | 0 | 00 | | j | 000010 | X | X | 1 | XX |   须注意：凡是无关项（X）在代码中一律是用0代替的，这样方便实现。  真值表（续2）   |  |  |  |  |  | | --- | --- | --- | --- | --- | | opcode | ALUOp | funct | 运算 | ALUControl | | lw | 00 | XXXXXX | 加 | 010 | | sw | 00 | XXXXXX | 加 | 010 | | beq | 01 | XXXXXX | 减 | 110 | | R型 | 10 | 100000 | 加 | 010 | | 100010 | 减 | 110 | | 100100 | 与 | 000 | | 100101 | 或 | 001 | | 101010 | SLT | 111 |   2 存储器(BlockMemory)  参数设置如下：  C:\Users\windows10\AppData\Local\Microsoft\Windows\INetCache\Content.Word\批注 2020-06-25 211131.png  须注意：Basis页中的Generate address interface with 32 bits选项在实验指导书中是不让点上的，但是如果不选中，那么生成的指令存储器的地址就不是32位了，故这里仍点选。 | | | | | | |
| 四、实验过程或算法（源程序）  下面仅展示重要代码，其余代码参考附带的项目文件。   1. *maindec代码：*   **module maindec(**  **input [5:0]opcode,**  **output reg [8:0]signals**  **);**  **always @(\*)**  **begin**  **case(opcode)**  **6'b000000:**  **begin**  **signals<=9'b000110010;**  **end**  **6'b100011:**  **begin**  **signals<=9'b101010000;**  **end**  **6'b101011:**  **begin**  **signals<=9'b011000000; //x11x00000**  **end**  **6'b000100:**  **begin**  **signals<=9'b000001001; //x00x01001**  **end**  **6'b001000:**  **begin**  **signals<=9'b001010000;**  **end**  **6'b000010:**  **begin**  **signals<=9'b000000100; //x0xx0x1xx**  **end**  **default:**  **signals<=9'h0;**  **endcase**  **end**  **endmodule**   1. *aludec代码：*   **module aludec(**  **input [1:0]ALUOp,**  **input [5:0]funct,**  **output reg [2:0]ALUControl**  **);**  **always @ (\*)**  **begin**  **case({ALUOp,funct})**  **8'b10100000:**  **begin**  **ALUControl<=3'b010;**  **end**  **8'b10100010:**  **begin**  **ALUControl<=3'b110;**  **end**  **8'b10100100:**  **begin**  **ALUControl<=3'b000;**  **end**  **8'b10100101:**  **begin**  **ALUControl<=3'b001;**  **end**  **8'b10101010:**  **begin**  **ALUControl<=3'b111;**  **end**  **default:**  **case(ALUOp)**  **2'b00:**  **begin**  **ALUControl<=3'b010;**  **end**  **2'b01:**  **begin**  **ALUControl<=3'b110;**  **end**  **default:**  **ALUControl<=3'b000;**  **endcase**  **endcase**  **end**  **endmodule**   1. *controller代码：*   **module controller(**  **input [5:0] opcode,**  **input [5:0] funct,**  **output MemtoReg,**  **output MemWrite,**  **output ALUSrc,**  **output RegDst,**  **output RegWrite,**  **output Branch,**  **output Jump,**  **output [2:0]ALUControl**  **//,output [1:0] test\_aluop**  **);**  **wire [31:0] signals;**  **wire [1:0]ALUOp;**  **maindec u1(opcode,signals);**  **assign ALUOp=signals[1:0];**  **aludec u2(ALUOp,funct,ALUControl);**  **assign MemtoReg=signals[8];**  **assign MemWrite=signals[7];**  **assign ALUSrc=signals[6];**  **assign RegDst=signals[5];**  **assign RegWrite=signals[4];**  **assign Branch=signals[3];**  **assign Jump=signals[2];**  **// assign test\_aluop=ALUOp;**  **endmodule**   1. *top代码：*   **module top(**  **input clk,**  **input rst,**  **output MemtoReg, MemWrite, ALUSrc, RegDst, RegWrite, Branch, Jump,**  **output wire [2:0]ALUControl,**  **output wire [6:0]seg,**  **output wire [7:0]ans**  **/\*, output wire [31:0]test\_instruction,**  **output wire [5:0]test\_op,**  **output wire [5:0]test\_funct,**  **output wire [1:0]test\_aluop\*/**  **);**  **reg [31:0]count=0;**  **reg clk0=0;**  **always @(posedge clk)**  **begin**  **if(count==32'd50000000)**  **begin**  **count<=0;**  **clk0<=~clk0;**  **end**  **else**  **count<=count+1;**  **end**  **wire [31:0] address;**  **wire [31:0] next\_address;**  **pc\_adder pc\_adder(address,next\_address);**  **wire ena;**  **pc pc(next\_address,clk0,rst,address,ena);**  **wire [31:0]instruction;**  **instruction\_rom instruction\_rom(**  **.clka(clk),**  **.ena(ena),**  **.addra(address),**  **.douta(instruction)**  **);**  **controller controller(instruction[31:26],instruction[5:0],**  **MemtoReg, MemWrite, ALUSrc, RegDst, RegWrite, Branch, Jump,ALUControl/\*,test\_aluop\*/);**  **display display(clk,rst,instruction,seg,ans);**  **/\*assign test\_instruction=instruction;**  **assign test\_op=instruction[31:26];**  **assign test\_funct=instruction[5:0];\*/**  **endmodule**  其中，红色部分为时钟分频代码，由于时钟频率太快，要保证显示在数码管上的指令出现的速度为1秒1个，只能分频。然而需要注意，display模块不能用分频后的时钟信号，应用原时钟信号。 | | | | | | |
| 五、实验结果及分析和（或）源程序调试过程  仿真结果如下：  现随机从仿真图中挑选三个指令，参考coe文件，可知这三个指令都在coe文件中，由上文提供的真值表可知，三个指令的控制信号都正确：  其中，test\_instruction表示指令（用16进制表示），变量名前加test表示这是中间变量，方便仿真时查看，在下板的时候需要注释掉。  下板结果如下：  C:\Users\windows10\AppData\Local\Microsoft\Windows\INetCache\Content.Word\TIM图片20200625221527.jpg  C:\Users\windows10\AppData\Local\Microsoft\Windows\INetCache\Content.Word\TIM图片20200625221507.jpg  图中3条指令为coe文件中第3、4、5条指令。  LED灯从最右至左分别为MemtoReg，MemWrite，ALUSrc，RegDst，RegWrite，Jump，Branch，ALUControl[0]，ALUControl[1]，ALUControl[2]。  最右边的开关为控制rst信号，用以实现刷新功能，刷新方式为同步刷新。 | | | | | | |