实验五 指令译码器设计与调试

**一、实验目的**

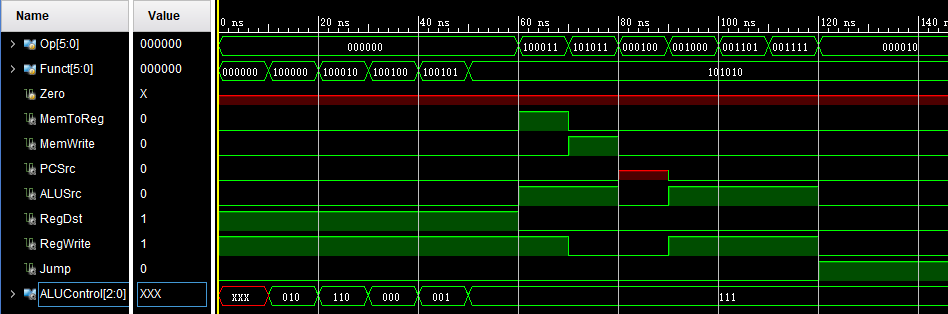
1. 学习并实现指令译码相关内容。

**二、实验内容**

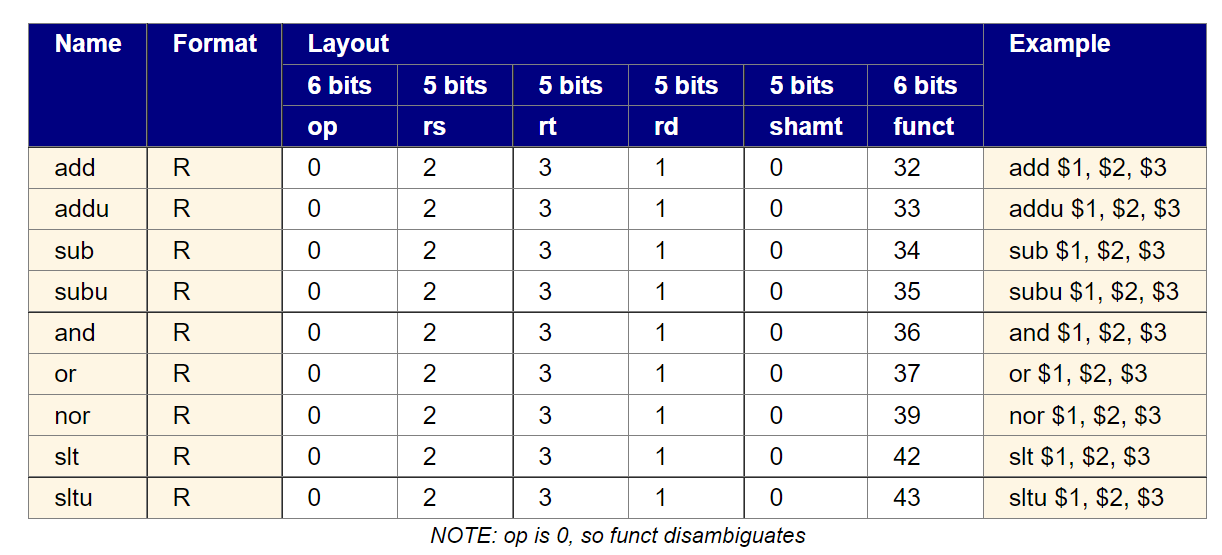
1. 设计一个指令译码器。
2. 译码器支持的指令集为：{addu, subu, ori, lw, sw, beq, lui, nop}。
3. nop机器码为0x00000000， 即空指令，不进行任何有效行为（修改寄存器等）。
4. 设计ALU译码器。
5. addu,subu可以不支持溢出。

**三、实验要求**

1. 通过真值表验证译码模块。



当Op为0时，对应的汇编指令为R类型指令：寄存器类型，故寄存器相关的RegDst和RegWrite均为一其余均为0，Funct的不同值分别对应了不同的汇编代码。见下图所示：



c，分别对应的汇编指令和对应控制位变化如下图真值表所示：



可见于仿真波形一一对应。

1. 记录设计和调试过程。

由于需要用到ALU模块功能，先对应ALU功能，编写ALU模块，对于需要用到ALU的汇编指令，令其产生ALU片选信号。对于不同需要ALU模块功能的汇编指令，更具其Funct值将其于ALU功能一一对应。对于不需要用到ALU的汇编代码，直接设置寄存器、内存等的使能信号即可。

**四、实验过程及结果**

Controller .v

module Controller(

input [5:0] Op, Funct,

input Zero,

output MemToReg, MemWrite,

output PCSrc, ALUSrc,

output RegDst, RegWrite,

output Jump,

output [2:0] ALUControl);

wire [1:0] ALUOp;

wire Branch;

MainDec MainDec\_1(Op, MemToReg, MemWrite, Branch, ALUSrc, RegDst, RegWrite, Jump, ALUOp);

ALUDec ALUDec\_1(Funct, ALUOp, ALUControl);

assign PCSrc = Branch & Zero;

endmodule

MainDec.v

module MainDec(

input [5:0] Op,

output MemToReg, MemWrite,

output Branch, ALUSrc,

output RegDst, RegWrite,

output Jump,

output [1:0] ALUOp);

reg [8:0] Controls;

assign {RegWrite, RegDst, ALUSrc, Branch, MemWrite, MemToReg, Jump, ALUOp} = Controls;

always@(\*)

case(Op)

6'b000000: Controls <= 9'b110000010;//register-type

6'b100011: Controls <= 9'b101001000;//LW

6'b101011: Controls <= 9'b001010000;//SW

6'b000100: Controls <= 9'b000100001;//BEQ

6'b001000: Controls <= 9'b101000000;//ADDI

6'b001101: Controls <= 9'b101000000;//ORI

6'b001111: Controls <= 9'b101000000;//LUI

6'b000010: Controls <= 9'b000000100;//J

default: Controls <= 9'bxxxxxxxxx;//illegal Op

endcase

endmodule

ALUDec.v

module ALUDec(

input [5:0] Funct,

input [5:0] ALUOp,

output reg [2:0] ALUControl);

always@(\*)

case(ALUOp)

2'b00: ALUControl <= 3'b010;//add (for lw/sw/addi)

2'b01: ALUControl <= 3'b110;//sub (for beq)

default :case(Funct) //R-type Instructions

6'b000000: ALUControl <= 3'bxxx;//nop

6'b100000: ALUControl <= 3'b010;//add

6'b100010: ALUControl <= 3'b110;//sub

6'b100100: ALUControl <= 3'b000;//and

6'b100101: ALUControl <= 3'b001;//or

6'b101010: ALUControl <= 3'b111;//slt

default: ALUControl <= 3'bxxx;//???

endcase

endcase

endmodule

ALU.v

module ALU(OP, A, B, F);

parameter SIZE = 32;

input [3:0] OP;

input [SIZE:1] A;

input [SIZE:1] B;

output reg [SIZE:1] F;

reg C;

always @(\*)

begin

C = 0;

case(OP)

4'b0000: begin F = A & B; end

4'b0001: begin F = A | B; end

4'b0010: begin {C, F} = A + B; end

4'b0011: begin F = ~(A | B); end

4'b0100: begin F = A ^ B; end

4'b0101: begin F = B << A; end

4'b0110: begin {C, F} = A - B; end

4'b0111: begin F = A < B; end

endcase

end

endmodule

sim\_Controller.v

module sim\_Controller();

reg [5:0] Op, Funct;

reg Zero;

wire MemToReg, MemWrite;

wire PCSrc, ALUSrc;

wire RegDst, RegWrite;

wire Jump;

wire [2:0] ALUControl;

initial begin

Op = 6'b000000;Funct = 6'b000000;

fork

begin

Op = 6'b000000;//register-type

Funct = 6'b000000;//nop

#10 Funct = 6'b100000;//add

#10 Funct = 6'b100010;//sub

#10 Funct = 6'b100100;//and

#10 Funct = 6'b100101;//or

#10 Funct = 6'b101010;//slt

Op = 6'b000000;//register-type

#10 Op = 6'b100011;//LW

#10 Op = 6'b101011;//SW

#10 Op = 6'b000100;//BEQ

#10 Op = 6'b001000;//ADDI

#10 Op = 6'b001101;//ORI

#10 Op = 6'b001111;//LUI

#10 Op = 6'b000010;//J

end

join

end

Controller Controller\_1(Op, Funct, Zero, MemToReg, MemWrite, PCSrc, ALUSrc, RegDst, RegWrite, Jump, ALUControl);

endmodule

**五、调试和心得体会**

不同功能说到底是由不同信号产生的，信号和功能一一对应，不同的汇编指令对应的就是一串二进制字符串，分析各位的数值即数值对应的信息，在代码中赋予不同信号就能产生不同功能。

封装先前写过的模块可以用于新模块中。