

COD LAB4 实验报告

姓名：刘芷辰

学号：PB21111728

日期：2023.5.15

1. 实验题目

单周期CPU

2. 实验目标

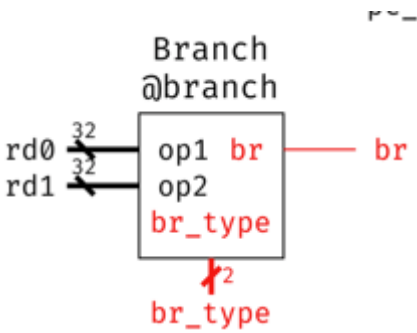
- 理解单周期CPU的结构和工作原理
- 熟练掌握单周期CPU数据通路和控制器的设计和描述方法
- 理解单周期CPU的调试方法

3. 实验内容

3.1 CPU设计与原理

branch模块

- 框图



- 代码

```
1 module Branch(  
2     input [31:0] op1,op2,
```

```

3      input [1:0] br_type,
4      output reg br
5  );
6      always @(*) begin
7          case (br_type)
8              // beq
9              2'b01: br = (op1 == op2) ? 1 : 0;
10
11              //blt
12              2'b10:
13                  if (op1[31] == op2[31]) begin
14                      if (op1[30:0] < op2[30:0])
15                          br = 1;
16                      else
17                          br = 0;
18                  end
19                  else if ( op1[31] == 1 ) br = 1;
20                  else br = 0;
21
22              default: br = 0;
23          endcase
24
25      end
26  endmodule

```

- 设计思路

brtype 2'b01表示beq, 2'b10表示blt

br=1表示跳转

br=0表示不需跳转

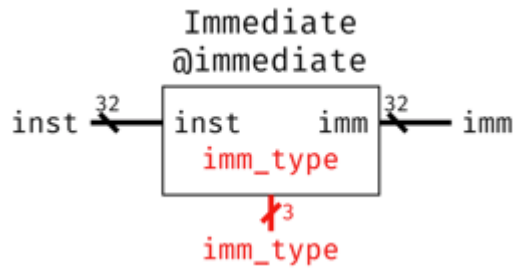
对于beq, 只需要判断op1和op2是否相等, 相等则跳转, br=1

对于blt, 首先比较符号位是否相同, 相同则比较后31位, op1小于op2则跳转, br=1

若符号位不相同要满足小于则只需要op1的符号位是1即可, 此时需要跳转, br=1, 其余情况为0

imm模块

- 框图



- 代码

```

1  module Immediate(
2      input [31:0] inst,
3      output reg [31:0] imm
4  );
5      always@(*) begin
6          case(inst[6:0])
7              //I-type
8              //addi
9              7'b0010011: imm = {{20{inst[31]}},inst[31:20]};
10             //lw
11             7'b0000011: imm = {{20{inst[31]}},inst[31:20]};
12             //jalr
13             7'b1100111: imm = {{20{inst[31]}},inst[31:20]};
14
15
16             //B-type
17             //beq/blt
18             7'b1100011: imm =
19                 {{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};
20
21
22             //S-type
23             //sw
24             7'b0100011: imm = {{20{inst[31]}},inst[31:25],inst[11:7]};
25
26
27             //U-type
28             //lui
29             7'b0110111: imm = {inst[31:12],12'b0};
30             //auipc
31             7'b0010111: imm = {inst[31:12],12'b0};
32
33
34             //J-type
35             //jal

```

```

35     7'b1101111: imm =
    {{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};
36
37     default: imm = 32'b0;
38     endcase
39 end
40 endmodule

```

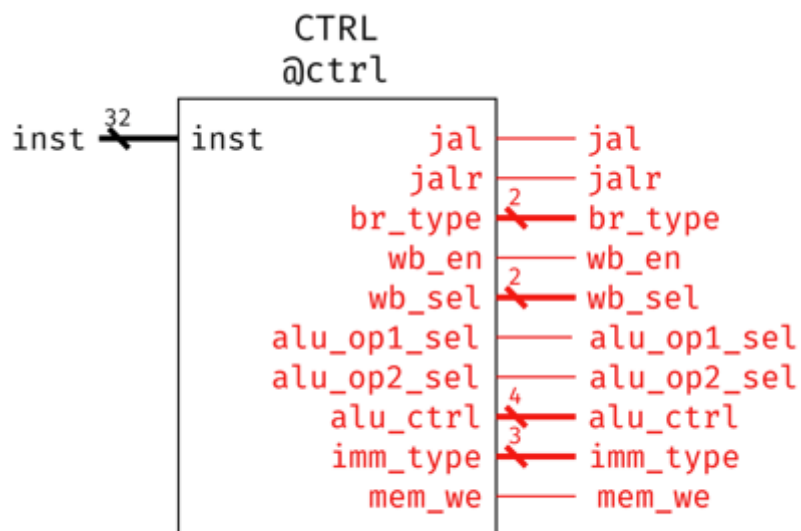
- 设计思路

31	27	26	25	24	20	19	15	14	12	11	7	6	0		
funct7				rs2		rs1	funct3		rd	opcode				R-type	
imm[11:0]						rs1	funct3		rd	opcode				I-type	
imm[11:5]				rs2		rs1	funct3		imm[4:0]		opcode			S-type	
imm[12:10:5]				rs2		rs1	funct3		imm[4:1 11]		opcode			B-type	
imm[31:12]										rd		opcode			U-type
imm[20 10:1 11 19:12]										rd		opcode			J-type

根据不同的指令类型对imm进行扩展

CTRL模块

- 框图



- 代码

```

1 module CTRL(
2     input [31:0] inst,
3     output reg jal,
4     output reg jalr,

```

```

5     output reg [1:0] br_type,
6     output reg wb_en,
7     output reg [1:0] wb_sel,
8     output reg alu_op1_sel,
9     output reg alu_op2_sel,
10    output reg [3:0] alu_ctrl,
11    output reg mem_we
12 );
13 always@(*) begin
14     case(inst[6:0])
15     //add
16     7'b0110011: begin
17         jal = 0;
18         jalr = 0;
19         br_type = 2'b00;
20         wb_en = 1;
21         wb_sel = 2'b00;
22         alu_op1_sel = 0;
23         alu_op2_sel = 0;
24         alu_ctrl = 0;
25         mem_we = 0;
26     end
27
28
29     //addi
30     7'b0010011: begin
31         jal = 0;
32         jalr = 0;
33         br_type = 2'b00;
34         wb_en = 1;
35         wb_sel = 2'b00;
36         alu_op1_sel = 0;
37         alu_op2_sel = 1;
38         alu_ctrl = 0;
39         mem_we = 0;
40     end
41     //jalr
42     7'b1100111: begin
43         jal = 0;
44         jalr = 1;
45         br_type = 2'b00;
46         wb_en = 1;
47         wb_sel = 2'b01;
48         alu_op1_sel = 0;
49         alu_op2_sel = 1;

```

```

50         alu_ctrl = 0;
51         mem_we = 0;
52     end
53     //lw
54     7'b0000011: begin
55         jal = 0;
56         jalr = 0;
57         br_type = 2'b00;
58         wb_en = 1;
59         wb_sel = 2'b10;
60         alu_op1_sel = 0;
61         alu_op2_sel = 1;
62         alu_ctrl = 0;
63         mem_we = 0;
64     end
65
66
67     //beq,blt
68     7'b1100011: begin
69         jal = 0;
70         jalr = 0;
71         if(inst[14:12] == 3'b000)
72             br_type = 2'b01;
73         else if(inst[14:12] == 3'b100)
74             br_type = 2'b10;
75         else br_type = 2'b00;
76         wb_en = 0;
77         wb_sel = 2'b00;
78         alu_op1_sel = 1;
79         alu_op2_sel = 1;
80         alu_ctrl = 0;
81         mem_we = 0;
82     end
83
84
85     //sw
86     7'b0100011: begin
87         jal = 0;
88         jalr = 0;
89         br_type = 2'b00;
90         wb_en = 0;
91         wb_sel = 2'b00;
92         alu_op1_sel = 0;
93         alu_op2_sel = 1;
94         alu_ctrl = 0;

```

```

95         mem_we = 1;
96     end
97
98
99     //lui
100    7'b0110111: begin
101        jal = 0;
102        jalr = 0;
103        br_type = 2'b00;
104        wb_en = 1;
105        wb_sel = 2'b11;
106        alu_op1_sel = 0;
107        alu_op2_sel = 0;
108        alu_ctrl = 4'b1000;
109        mem_we = 0;
110    end
111    //auipc
112    7'b0010111: begin
113        jal = 0;
114        jalr = 0;
115        br_type = 2'b00;
116        wb_en = 1;
117        wb_sel = 2'b00;
118        alu_op1_sel = 1;
119        alu_op2_sel = 1;
120        alu_ctrl = 0;
121        mem_we = 0;
122    end
123
124
125    //jal
126    7'b1101111: begin
127        jal = 1;
128        jalr = 0;
129        br_type = 2'b00;
130        wb_en = 1;
131        wb_sel = 2'b01;
132        alu_op1_sel = 1;
133        alu_op2_sel = 1;
134        alu_ctrl = 0;
135        mem_we = 0;
136    end
137
138
139    default: begin

```

```

140         jal = 0;
141         jalr = 0;
142         br_type = 2'b11;
143         wb_en = 0;
144         wb_sel = 2'b00;
145         alu_op1_sel = 0;
146         alu_op2_sel = 0;
147         alu_ctrl = 10;
148         mem_we = 0;
149     end
150 endcase
151 end
152 endmodule

```

- 设计原理

- add: 不产生jar、jalr、br_type信号，结果来自alu_res所以wb_sel为0，结果写入rd所以wb_en为1，相加所以alu_ctrl选择0，相加的数都来自寄存器所以两个alu_op_sel均为0，不需要存入存储器所以mem_we为0

- addi:

✓ `addi rd, rs1, imm` # $x[rd] = x[rs1] + \text{sext}(imm)$

不产生jar、jalr、br_type信号，结果来自alu_res所以wb_sel为0，结果写入rd所以wb_en为1，相加所以alu_ctrl选择0，相加的数都一个来自寄存器一个来自imm所以两个alu_op_sel一个为0，一个为1，不需要存入存储器所以mem_we为0

- jalr:

✓ `jalr rd, offset(rs1)` # $t = pc + 4; pc = (x[rs1] + \text{sext}(\text{offset})) \& \sim 1; x[rd] = t$

jalr为1，其余跳转信号为0，选择pc_add4作为写入rd的数据所以wb_sel为1，wb_en为1，跳转的pc是寄存器内容和imm相加，所以alu_ctrl选择0，alu_op_sel一个为0，一个为1，不需要存入存储器所以mem_we为0

- lw:

✓ `lw rd, offset(rs1)` # $x[rd] = M[x[rs1] + \text{sext}(\text{offset})]$

不产生jar、jalr、br_type信号，存入rd的数据来自mem_rd,所以wb_sel为2，wb_en为1，地址为寄存器内容加上偏移量，所以alu_ctrl选择0，alu_op_sel一个为0，一个为1，不需要存入存储器所以mem_we为0

- beq, blt

✓ beq rs1, rs2, offset # if (rs1 == rs2) pc += sext(offset)

✓ blt rs1, rs2, offset # if (rs1 <_s rs2) pc += sext(offset)

br_type信号根据inst[14:12]来确定，其余跳转信号为0，不写入寄存器所以wb_sel、wb_en为0，Branch模块得到br信号，br=1 就会选择 alu_res为下一个pc，pc+偏移量则alu_op_sel都为1，alu_ctrl选择0，不需要存入存储器所以mem_we为0

- sw

✓ sw rs2, offset(rs1) # M[x[rs1] + sext(offset)] = x[rs2]

不产生jar、jalr、br_type信号，不写入寄存器所以wb_sel、wb_en为0，地址为寄存器内容加上偏移量，所以alu_ctrl选择0,alu_op_sel一个为0，一个为1，需要存入存储器所以mem_we为1

- lui

✓ lui rd, imm # x[rd] = imm[31:12] << 12

不产生jar、jalr、br_type信号，将imm写入rd，则wb_en为1，wb_sel为3，左移12位但不是通过alu实现而是imm模块，因此alu_ctrl都可以，alu_op_sel均为0，不需要存入存储器所以mem_we为0

- auipc

✓ auipc rd, imm # x[rd] = pc + imm[31:12] << 12

不产生jar、jalr、br_type信号，pc+imm存入rd，所以wb_sel为0，wb_en为1，同B-type，alu_op_sel都为1，alu_ctrl选择0，不需要存入存储器所以mem_we为0

- jal

✓ jal rd, offset # x[rd] = pc+4, pc += sext(offset)

jal为1，其余跳转信号为0，pc_add4作为写入rd的数据所以wb_sel为1，wb_en为1，同auipc，alu_op_sel都为1，alu_ctrl选择0，不需要存入存储器所以mem_we为0

3.2 实验结果

test

- 仿真

4. 分析数据通路差异

在`pc_next`的处理上有区别，在各类跳转指令比如`jal`、`jalr`、`beq`、`blt`上体现

5. 总结

本次实验难度适中

通过单周期CPU的设计，更加了解了计算机体系结构，对riscv指令集的架构理解更加深入

对理解较为复杂的`verilog`代码之间的联系很有帮助