《计算机组成原理》 实验报告



实验题目:多周期 CPU

学生姓名: 王志强

学生学号: PB18051049

完成日期: 2020.05.25

计算机实验教学中心制 2019年9月

一、实验目标

- 理解计算机硬件的基本组成、结构和工作原理;
- 掌握数字系统的设计和调试方法;
- 熟练掌握数据通路和控制器的设计和描述方法。

二、实验内容

1. 多周期CPU

待设计的多周期CPU可以执行如下6条指令:

- R类指令
 - o add: rd <- rs + rt** **op = 000000, funct = 100000

op(6 bit	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)
----------	------------	------------	------------	---------------	---------------

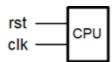
- |类指令
 - addi: rt <- rs+imm op = 001000
 lw: rt <- M(rs+addr) op = 100011
 sw: M(rs+addr) <- rt op = 101011
 - \circ beq: if(rs == rt) then pc <-pc+4+addr<<2; else pc <- pc+4 op = 000100

op(6 bits) rs(5 bits) rt(5 bits)	addr/immediate(16 bits)
----------------------------------	-------------------------

- |类指令
 - o j: pc <- (pc+4)[31:28] | (add<<2)[27:0] op = 000010

```
op(6 bits) addr(26 bits)
```

待设计的CPU逻辑符号和端口声明如下:

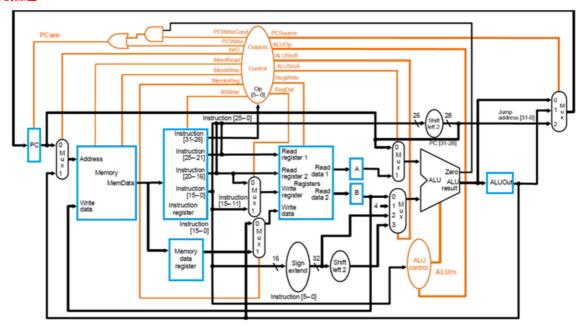


```
1 module cpu_one_cycle( //单周期CPU
2 input clk, //时钟(上升沿有效)
3 input rst //异步复位,高电平有效
4 );
5 ......
6 endmodule
```

1.1 数据通路

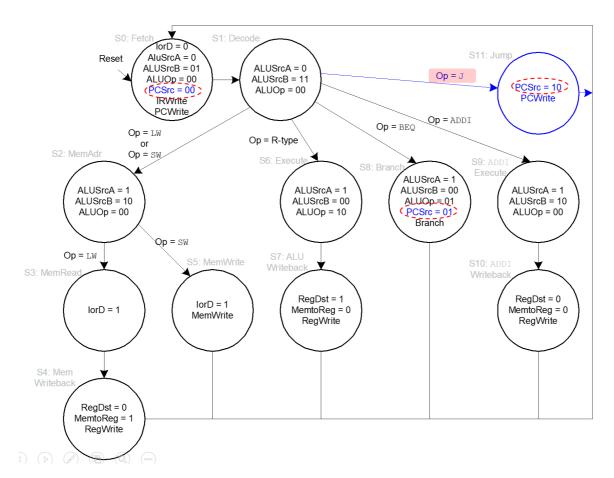
满足上述指令的功能,设计多周期CPU的数据通路和控制器(橙色部分)如下图所示。具体实现时ALU和寄存器堆可以利用实验1和实验2设计的模块,指令和数据存储共用一个RAM存储器,采用IP例化实现,容量为512 x 32位的分布式存储器。

注意: ALU control输出信号对应lab1中的alu控制信号,移位和符号拓展模块直接在顶层模块中进行行为描述



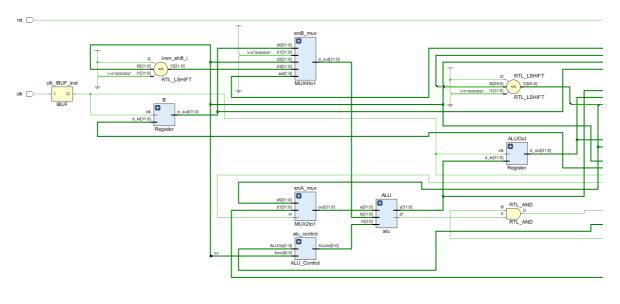
1.2 控制器状态图 (补充addi指令)

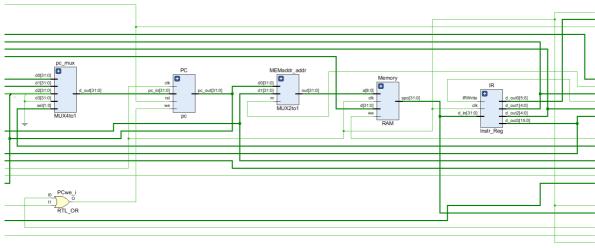
根据各类指令在每个阶段完成的工作,设计控制器状态图,如下

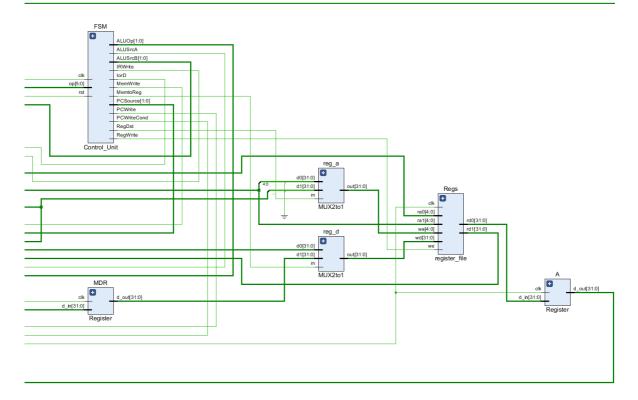


1.3 RTL ANALYSIS-Schematic

从上至下左右拼接







1.4 核心模块Verilog实现

1.4.1 多周期CPU顶层模块

```
module cpu(
1
2
        input clk,rst
3
        );
4
        wire [31:0] pc_cur,pc_next,pc_jump; //pc
5
6
        wire [31:0] MEMdata; //取出的指令或者数据
        wire [31:0] MEM_addr;
7
8
        wire [5:0] IR_op;
                             //op,传给control unit
9
        wire [4:0] IR_rs,IR_rt; //rs,rt
10
        wire [15:0] IR_Imm; //指令后16位
        wire [31:0] MDR_out; //MDR输出数据
11
        wire [4:0] Reg_addr; //寄存器堆写入地址
12
13
        wire [31:0] Reg_data; //寄存器堆写入数据
        wire [31:0] Imm_signext,Imm_shift; //符号拓展后的立即数,左移后的立即数
14
        wire [31:0] rdA,rdB;
15
                              //寄存器堆两端口输出数据
        wire [31:0] regA, regB; //A, B寄存器端口读出数据
16
17
        wire [31:0] srcA, srcB, ALU_result; //ALU操作数,结果
                          //ALU控制信号
        wire [2 :0] ALUm;
18
19
        wire [31:0] ALUout;
                               //ALUOut寄存器输出
        wire ALUzero, PCwe; //alu零输出标志, PC寄存器写使能
20
21
        /*下面为控制信号*/
22
       wire PCWriteCond, PCWrite, IorD, MemRead, MemWrite; //MemRead暂时不用
23
        wire MemtoReg,IRWrite,RegDst,RegWrite,ALUSrcA;
24
        wire [1:0] ALUSrcB, ALUOp, PCSource;
25
        //data path
26
27
        assign PCwe = PCWrite(PCWriteCond&ALUzero);
        pc PC(
28
29
            .clk
                            (c1k
                                            ),
30
            .rst
                            (rst
                                            ),
31
            .we
                            (PCwe
                                            ),
                                            ),
32
            .pc_in
                            (pc_next
33
            .pc_out
                            (pc_cur
                                            )
34
        ); //PC
35
        MUX2to1 MEMaddr_addr(
36
37
                                            ),
            . m
                            (IorD
            .d0
38
                                            ),
                            (pc_cur
39
            .d1
                            (ALUout
                                            ),
40
                            (MEM_addr
            .out
41
        );
42
43
        RAM Memory(
44
            .clk
                            (clk
                                            ),
                            (MemWrite
45
            .we
                                            ),
46
            .a
                            (MEM_addr[10:2]),
47
            . d
                            (regB
                                            ),
48
                            (MEMdata
                                            )
            .spo
49
        );
50
51
        Instr_Reg IR(
52
                            (c1k
                                            ),
            .clk
53
                                            ),
            .IRWrite
                            (IRWrite
54
            .d_in
                                            ), //instr
                            (MEMdata
55
            .d_out0
                            (IR_op
                                               //op
```

```
56
               .d\_out1
                                                  ), //rs
                                (IR_rs
 57
               .d_out2
                                (IR_rt
                                                  ), //rt
                                                      //末16位
 58
               .d_out3
                                (IR_Imm
 59
          ); //IR
 60
 61
          Register MDR(
 62
              .clk
                                (clk
                                                  ),
               .d_in
 63
                                (MEMdata
                                                  ),
 64
               .d_out
                                (MDR_out
                                                  )
 65
               //MDR
 66
 67
          MUX2to1 reg_a(
 68
               . m
                                                  ),
                                (RegDst
 69
               .d0
                                (IR_rt
                                                  ),
 70
               .d1
                                (IR_Imm[15:11]
                                                 ),
 71
               .out
                                (Reg_addr
                                                  )
 72
          ); //write rt or rd
 73
 74
          MUX2to1 reg_d(
 75
               . m
                                (MemtoReg
                                                  ),
               .d0
 76
                                (ALUout
                                                  ),
 77
               .d1
                                (MDR_out
                                                  ),
 78
               .out
                                (Reg_data
                                                  )
 79
          );
 80
 81
          register_file Regs(
 82
               .clk
                                                  ),
                                (clk
              .ra0
 83
                                                  ),
                                (IR_rs
               .rd0
                                                  ),
 84
                                (rdA
                                                  ),
 85
               .ra1
                                (IR_rt
               .rd1
                                (rdB
 86
                                                  ),
 87
               .we
                                (RegWrite
                                                  ),
 88
                                                  ),
               .wa
                                (Reg_addr
                                                  )
 89
              .wd
                                (Reg_data
 90
          ); //register file
 91
 92
          Register A(
 93
               .clk
                                (c1k
                                                  ),
 94
               .d_in
                                                  ),
                                (rdA
 95
               .d\_out
                                (regA
                                                  )
 96
          );
 97
          Register B(
 98
99
               .clk
                                                  ),
                                (clk
               .d_in
100
                                (rdB
                                                  ),
101
               .d_out
                                (regB
                                                  )
102
          );
103
104
          MUX2to1 srcA_mux(
105
               . m
                                (ALUSTCA
                                                  ),
106
               .d0
                                                  ),
                                (pc_cur
107
               .d1
                                (regA
                                                  ),
108
                                                  )
              .out
                                (srcA
109
          );
110
          assign Imm_signext = {{16{IR_Imm[15]}},IR_Imm}; //sign extend
111
112
          assign Imm_shift = (Imm_signext<<2);</pre>
                                                      //shift left2
113
          MUX4to1 srcB_mux(
```

```
114
               .sel
                                (ALUSrcB
                                                  ),
                                                  ),
115
               .d0
                                (regB
116
               .d1
                                (32'd4
                                                  ),
               . d2
117
                                (Imm_signext
                                                  ),
               .d3
118
                                (Imm_shift
                                                  ),
119
               .d_out
                                (srcB
                                                  )
120
          );
121
122
          alu ALU(
123
               .a
                                (srcA
                                                  ),
124
               .b
                                (srcB
                                                  ),
125
                                (ALUm
                                                  ),
               . m
126
               . у
                                (ALU_result
                                                  ),
127
               .zf
                                (ALUzero
                                                  )
128
          ); //alu
129
130
          Register ALUOut(
131
               .clk
                                (clk
                                                  ),
132
               .d_in
                                (ALU_result
                                                  ),
133
               .d_out
                                (ALUout
                                                  )
134
          );
135
136
137
          assign pc_jump = {pc_cur[31:28],{{IR_rs,IR_rt,IR_Imm}<<2}};</pre>
138
          MUX4to1 pc_mux(
139
               .sel
                                (PCSource
                                                  ),
140
               .d0
                                                  ),
                                (ALU_result
                                (ALUout
141
               .d1
                                                  ),
142
               . d2
                                                  ),
                                (pc_jump
143
               .d3
                                (32'd0
                                                  ),
144
               .d_out
                                                  )
                                (pc_next
145
          );
146
147
          //FSM,Control Unit
148
          Control_Unit FSM(
149
               .clk
                                (clk
                                                  ),
150
               .rst
                                (rst
                                                  ),
151
               .op
                                (IR_op
                                                  ),
152
               .PCWriteCond
                                (PCWriteCond
                                                  ),
153
               .PCWrite
                                (PCWrite
                                                  ),
154
               .IorD
                                (IorD
                                                  ).
155
               .MemRead
                                (MemRead
                                                  ),
                                                  ),
156
               .MemWrite
                                (MemWrite
157
               .MemtoReg
                                (MemtoReg
                                                  ),
158
               .IRWrite
                                (IRWrite
                                                  ),
159
                                                  ),
               .RegDst
                                (RegDst
160
               .RegWrite
                                (RegWrite
                                                  ),
161
               .ALUSrcA
                                (ALUSTCA
                                                  ),
                                                  ),
162
               .ALUSrcB
                                (ALUSTCB
163
               .ALUOp
                                (ALUOp
                                                  ),
164
                                (PCSource
                                                  )
               .PCSource
165
          );
166
167
          ALU_Control alu_control(
168
               .ALUOp
                                (ALUOp
                                                  ),
                                                  ),
169
               .funct
                                (IR_Imm[5:0]
170
               .ALUm
                                (ALUm
                                                  )
171
          );
```

• 核心模块control unit实现

```
1
    module Control_Unit(
 2
        input [5:0] op,
 3
        output
    RegDst,Jump,Branch,MemtoReg,MemRead,MemWrite,ALUSrc,RegWrite,
4
        output [2:0] ALUOp //待实现的指令有限,故直接生成alu控制信号
 5
        );
 6
        //实现以下6条指令
 7
        parameter add = 6'b000000;
 8
        parameter addi = 6'b001000;
9
        parameter lw = 6'b100011;
        parameter sw = 6'b101011;
10
11
        parameter beq = 6'b000100;
                       = 6'b000010;
12
        parameter j
13
        //add_op = 000, sub_op = 000
14
        reg [10:0] control;
15
        assign {RegDst,Jump,Branch,MemtoReg,MemRead,
16
                MemWrite,ALUSrc,RegWrite,ALUOp} = control;
17
18
        always @(op)
19
        begin
20
            case(op)
21
            add : control = 11'b10000001000;
22
            addi: control = 11'b00000011000;
23
            lw : control = 11'b00011011000;
            sw : control = 11'bx00x0110000;
24
25
            beg : control = 11'bx01x0000001;
            j : control = 11'bx1xx00x0xxx;
26
27
            default: control = 11'bxxxxxxxxxx;
28
            endcase
29
        end
    endmodule
30
```

1.4.2 控制器状态机模块

```
1
    module Control_Unit(
 2
        input clk, rst,
 3
        input [5:0] op,
 4
        output PCWriteCond, PCWrite, IorD, MemRead, MemWrite,
 5
        output MemtoReg, IRWrite, RegDst, RegWrite, ALUSrcA,
 6
        output [1:0] ALUSrcB,ALUOp,PCSource
 7
        );
 8
        //instr parameter
 9
        parameter add = 6'b000000;
        parameter addi = 6'b001000;
10
11
        parameter lw = 6'b100011;
12
        parameter sw = 6'b101011;
13
        parameter beq = 6'b000100;
14
        parameter j
                     = 6'b000010;
15
16
        //state parameter
17
                             = 4'b0000;
        parameter IF
18
        parameter ID
                             = 4'b0001;
```

```
parameter LS_EX = 4'b0010;
19
20
        parameter LW_MEM
                             = 4'b0011;
21
                             = 4'b0100;
        parameter LW_WB
22
        parameter SW_MEM
                             = 4'b0101;
23
        parameter R_EX
                             = 4'b0110;
24
                             = 4'b0111;
        parameter R_WB
25
        parameter BEQ_EX
                             = 4'b1000;
26
        parameter J_EX
                             = 4'b1001;
27
                             = 4'b1010;
        parameter I_EX
28
        parameter I_WB
                             = 4'b1011;
29
30
        reg [3:0] cur_state,next_state;
31
        reg [15:0] control;
32
33
        assign {PCWriteCond, PCWrite, IorD, MemRead, MemWrite, MemtoReg, IRWrite,
34
                 RegDst,RegWrite,ALUSrcA,ALUSrcB,ALUOp,PCSource}=control;
35
        //三段式描述状态机
36
        always @(posedge clk,posedge rst)
37
        begin
38
            if(rst)
39
                 cur_state <= IF;</pre>
40
            else
41
                 cur_state <= next_state;</pre>
42
        end
43
        //状态转移条件
44
        always @(*)
45
        begin
46
            case(cur_state)
47
            IF:
                        next_state = ID;
            ID:
49
                 case(op)
50
                 add:
                         next_state = R_EX;
51
                 addi:
                         next_state = I_EX;
52
                 lw,sw: next_state = LS_EX;
53
                 beq:
                         next_state = BEQ_EX;
54
                         next_state = J_EX;
                 j:
55
                 default:next_state = 4'b0000;
56
                 endcase
57
            LS_EX:
58
                 case(op)
59
                 lw:
                         next_state = LW_MEM;
60
                         next_state = SW_MEM;
61
                 default:next_state = 4'b0000;
62
                 endcase
63
            LW_MEM:
                         next_state = LW_WB;
64
            R_EX:
                         next_state = R_WB;
65
            I_EX:
                         next_state = I_WB;
66
            LW_WB, SW_MEM, R_WB, I_WB, BEQ_EX, J_EX:
67
                         next_state = IF;
68
            default:
                         next_state = 4'b0000;
69
             endcase
70
        end
71
        //状态输出
72
        always @(cur_state)
73
        begin
74
            case(cur_state)
75
            IF:
                     control = 16'b0101001000010000;
76
                     control = 16'b000000000110000;
```

```
LS_EX: control = 16'b000000001100000;
77
78
           LW_MEM: control = 16'b001100000000000;
79
           LW_WB: control = 16'b00000100100000000;
80
           SW_MEM: control = 16'b0010100000000000;
81
           R_{EX}: control = 16'b000000001001000;
82
           R_WB: control = 16'b0000000110000000;
83
           I_EX: control = 16'b000000001100000;
84
           I_WB: control = 16'b0000000100000000;
85
           BEQ_EX: control = 16'b100000001000101;
86
                  control = 16'b0100000000000010;
87
           88
           endcase
89
       end
   endmodule
90
```

1.5 波形仿真

1.5.1仿真MIPS指令

```
# 初始PC = 0x00000000
1
2
 3
       j _start
                   # 0
4
 5
    .data
6
        .word 0,6,0,8,0x80000000,0x80000100,0x100,5,0,0,0
7
       #编译成机器码时,编译器会在前面多加个0,所以后面1w指令地址会多加4
8
9
    _start:
10
           addi $t0,$0,3
                                  \#t0=3
                                          44
                                          48
11
           addi $t1,$0,5
                                  \#t1=5
12
           addi $t2,$0,1
                                  #t2=1
                                          52
13
           addi $t3,$0,0
                                  #t3=0
14
15
           add $s0,$t1,$t0
                                  #s0=t1+t0=8 测试add指令
                                                            60
                                                            64
16
           lw $s1,12($0)
                                  #
17
           beq $s1,$s0,_next1
                                 #正确跳到_next
                                                            68
18
19
           j_fail
20
21
    _next1:
22
           lw $t0, 16($0)
                                 #t0 = 0x80000000
                                                     76
23
           lw $t1, 20($0)
                                  #t1 = 0 \times 80000100
                                                     80
24
25
           add $s0,$t1,$t0
                                  #s0 = 0x00000100 = 256 84
26
           lw $s1, 24($0)
                                  #
                                                         88
27
           beq $s1,$s0,_next2
                                 #正确跳到_success
                                                        92
28
29
           j_fail
30
31
    _next2:
32
           add $0, $0, $t2
                                                        100
                                  #$0应该一直为0
33
           beq $0,$t3,_success
                                                         104
34
35
36
   _fail:
37
                $t3,8($0) #失败通过看存储器地址0x08里值,若为0则测试不通过,最初地址
           SW
    0x08里值为0 108
```

```
      38
      j _fail

      39
      40

      40
      _success:

      41
      sw $t2,8($0) #全部测试通过,存储器地址0x08里值为1 116

      42
      j _success

      43
      #判断测试通过的条件是最后存储器地址0x08里值为1,说明全部通过测试
```

1.5.2 cpu仿真波形及解释

根据上述汇编指令,10个周期一组,仿真波形如下,cpu内部变量在各阶段的值在图中能看到,不一一解释

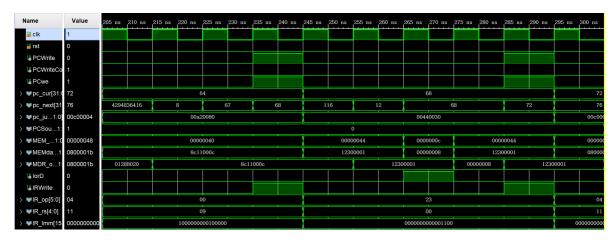
1.5.2.1 第一组 (1-10 cycle)

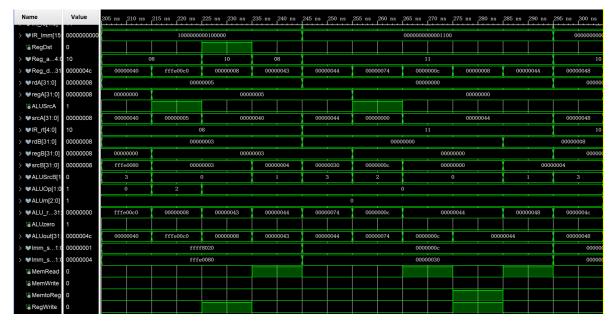


1.5.2.2 第二组 (11-20 cycle)



1.5.2.3 第三组 (21-30 cycle)





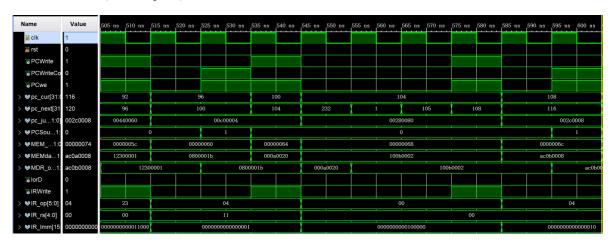
1.5.2.4 第四组 (31-40 cycle)

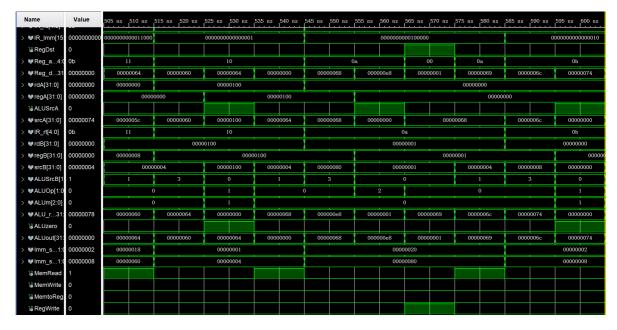


1.5.2.5 第五组 (41-50 cycle)

Name	Value			400	.05		105			450	455		*05	100	.05		105		105	500	
₩ clk	1	405 ns 410	s 415 hs	420 ns	425 ns	430 ns	435 ns	440 ns	445 ns	450 ns	400 ns	460 пѕ	400 ns	470 ns	4/5 ns	480 ns	485 ns	490 ns	495 ns	500 ns	
₩ rst	0											_									
1 PCWrite	1																				
	0																				
1⊌ PCwe	1																				
> W pc_cur[31:0	92		84		_				8				_				92				
> W pc_next[31		89	Y	88	42948	336440	2	56	21474	83736		92	1	88	γ	24	Y	1	00		
> № pc_ju1:0]	00440060	00240050		00a20080						Λ		00440060									
> ₩ PCSou1:	0																				
> ₩ MEM1:0	0000005c	00000054		00000058						<u> </u>	0000	0005c		00000018 00000							
> W MEMda1	12300001	01288020		\uparrow			8c11	0018				1	1230	00001		0000	00100	1	1230		
> W MDR_o1:	12300001	80000100 0128		88020 8c110018						4		Υ	1230	00001 00000100							
1⊌ IorD	0		7				1														
1⊌ IRWrite	1																				
> W IR_op[5:0]	23		23		Y			(0				Y				23				
> WIR_rs[4:0]	00		00	-	1				9				†		00						
> W IR_Imm[15	0000000000	0000	0000001010	0	1			10000000	00100000				00000000011000								
					,								-Ti-					•			
Name	Value	405 ns 410 r	s 415 ns	420 ns	425 ns	430 ns	435 ns	440 ns	445 ns	450 ns	455 ns	460 ns	465 ns	470 ns	475 ns	480 ns	485 ns	490 ns	495 ns	500 ns	
> W IR_Imm[15	0000000000	00000	0000001010	0	1			10000000	00100000			-				0	000000000	0011000			
1⊌ RegDst	0				1																
> W Reg_a4:0	11		09		1		08		1	0	(08					11				
> № Reg_d31	00000064	80000100	000	000059	0000	00058	fffe	8b00:	0000	0100	8000	00058	0000	005c	0000	000bc	0000	00018	0000	0100	
> W rdA[31:0]	00000000		0000000		80000100						00000000										
> W regA[31:0]	00000000		000	000000	80000100					00000000											
18 ALUSrcA	0																				
> W srcA[31:0]	0000005c		0000054		0000	00058	8000	0100		0000	0058		0000	1005c	0000	00000			000000)5c	
> W IR_rt[4:0]	11		09			08						11									
> W rdB[31:0]	00000100	00000005	800	000100		8000000						00000008									
> W regB[31:0]	80000000		0000005		80000100 80000000					00000008											
> W srcB[31:0]	00000004	00000005 00000004		fffe0080 80000000			0000	00004	0000	0060	00000018				00000008						
	0000004	00000005	000	000004	fffe	.0000	A						A			30010		0			
> WALUSTOB[1	1	00000005	000	1		3	\uparrow	(1			i		3		2			0		
> WALUSrcB[1 > WALUOp[1:0	1		000										1	3					0		
> W ALUOp[1:0	1		000	i							0		1	3	ſ				0		
> WALUOp[1:0 > WALUM[2:0]	1 0 0			i	1		1		8000	0058	0		1		. 0				00064		
> WALUOp[1:0 > WALUM[2:0]	1 0 0	0		0	1	3	1	2		0058	0	ı			. 0	2					
> WALUOp[1:0 > WALUm[2:0] > WALU_r31:	1 0 0 0 00000060	00000059		0	fffe	3	0000	2			0000	ı		000bc	0000	2	0000	0000		00000	
> WALUOp[1:0 > WALUm[2:0] > WALU_r31: 18 ALUzero	1 0 0 00000060 0	00000059	000	0	fffe	3 :00d8	0000	2	8000		0000	1 0005c	0000	000bc	0000	2	00000	0000		00000	
> % ALUOp[1:0 > % ALUm[2:0] > % ALU_r31:	1 0 0 00000060 0 00000064 00000018	00000059	0000059	0	fffe	3 :00d8	0000	200100	8000 0000 8020		0000	1 0005c	0000	000bc	0000	2	1	0000 00018		00000	
> WALUOp[1:0] > WALUm[2:0] > WALU_r31: 16 ALUzero > WALUout[31: > WImm_s1:0	1 0 0 00000060 0 00000064 00000018	00000059	0000059	0	fffe	3 :00d8	0000	00100 00100 00048	8000 0000 8020		0000	1 0005c	0000	000bc	0000	2	000000	0000 00018		00000	
> ₩ALUOp[1:0 > ₩ALUm[2:0] > ₩ALUm[2:0] > ₩ALU_r31: 18 ALUzero > ₩ALUout[31: > ₩Imm_s1:0 18 MemRead	1 0 0 00000060 0 00000064 00000018 00000060	00000059	0000059	0	fffe	3 :00d8	0000	00100 00100 00048	8000 0000 8020		0000	1 0005c	0000	000bc	0000	2	000000	0000 00018		00000	
> ₩ALUOp[1:0 > ₩ALUm[2:0] > ₩ALUm[2:0] > ₩ALU_r31: 18 ALUzero > ₩ALUout[31: > ₩Imm_s1:0 18 MemRead	1 0 0 00000060 0 00000064 00000018 00000060 1 0	00000059	0000059	0	fffe	3 :00d8	0000	00100 00100 00048	8000 0000 8020		0000	1 0005c	0000	000bc	0000	2	000000	0000 00018		00000	

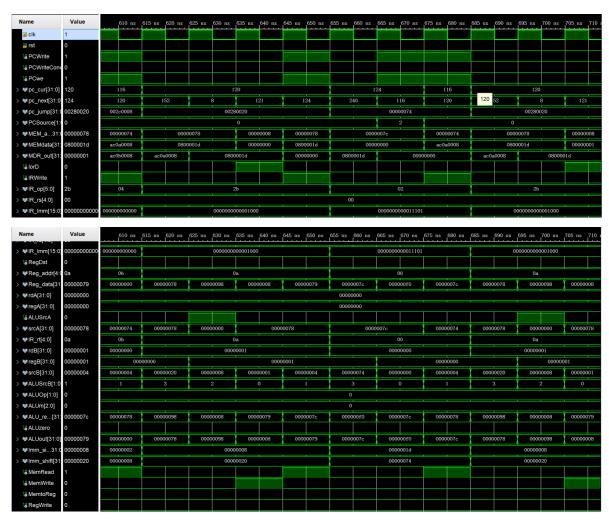
1.5.2.6 第六组 (51-60 cycle)





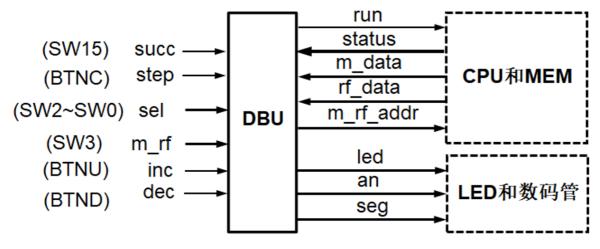
1.5.2.7 第七组 (61-71cycle)

最后在sw和j指令循环,可以看到第71个周期,MEM_addr地址为x01,MDR_OUT输出为1,即代表内存x08的值为1,测试通过



2、调试单元Debug Unit(DBU)

为了方便下载调试,设计一个调试单元DBU,该单元可以用于控制CPU的运行方式,显示运行过程的中间状态和最终运行结果。DBU的端口与CPU以及FPGA开发板外设(拨动/按钮开关、LED指示灯、7-段数码管)的连接如下图所示。为了DBU在不影响CPU运行的情况下,随时监视CPU运行过程中寄存器堆和数据存储器的内容,可以为寄存器堆和数据存储器增加1个用于调试的读端口。

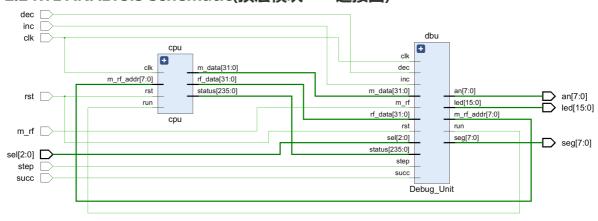


注: 图中省略了clk和rst信号

2.1 控制CPU运行方式

- succ = 1: clkd输出连续的周期性脉冲信号,可以作为CPU的时钟信号,控制CPU连续执行指令
- succ = 0:每按动step一次,clkd输出一个脉冲信号,可以作为CPU的时钟信号,控制CPU执行一个时钟周期
- sel = 0: 查看CPU运行结果 (存储器或者寄存器堆内容)
 - m_rf: 1, 查看存储器(MEM); 0, 查看寄存器堆(RF)
 - o m_rf_addr: MEM/RF的调试读口地址(字地址), 复位时为零
 - o inc/dec: m_rf_addr加1或减1
 - ∘ rf_data/m_data: 从RF/MEM读取的数据字
 - 16个LED指示灯显示m_rf_addr
 - 8个数码管显示rf_data/m_data
- sel = 1 ~ 7: 查看CPU运行状态 (status)
- 16个LED指示灯(SW15~SW0)依次显示控制器的控制信号PCSource(2)、PCwe、IorD、MemWrite、IRWrite、RegDst、MemtoReg、RegWrite、ALUm(3)、ALUSrcA、ALUSrcB(2)和ALUZero
- 8个数码管显示由sel选择的一个32位数据
 - o sel = 1: PC, 程序计数器
- sel = 2: IR, 指令寄存器
 - o sel = 3: MD, 存储器读出数据寄存器
- sel = 4: A, 寄存器堆读出寄存器A
 - sel = 5: B, 寄存器堆读出寄存器B
- sel = 6: ALUOut, ALU运算结果寄存器
 - \circ sel = 7:

2.2 RTL ANALYSIS-Schematic(顶层模块——连接图)



2.3文件架构



2.4 核心模块Verilog实现

2.4.1 顶层模块

```
1
      module dbu_top(
 2
          input clk,rst, //时钟, 复位
 3
          input succ,step,
                            //连续执行,单步执行
4
          input m_rf,inc,dec, //M/R选择, 地址加减
 5
          input [2:0] sel,
                            //输出控制
          output [15:0] led, //led
 6
 7
          output [7:0] an,
                            //seven segment enable
          output [7:0] seg //seven segment output
8
9
          );
10
11
          wire run;
12
          wire [207:0] status;
13
          wire [31:0] m_data,rf_data;
14
          wire [7:0] m_rf_addr;
15
        //data path
```

```
16
           Debug_Unit dbu(
17
                .clk
                                 (c1k
                                              ),
18
                                              ),
                .rst
                                 (rst
19
                .succ
                                              ),
                                 (succ
20
                .step
                                 (step
                                              ),
21
                .sel
                                 (sel
                                              ),
22
                .m_rf
                                 (m_rf
                                              ),
23
                .inc
                                 (inc
                                              ),
24
                .dec
                                 (dec
                                              ),
25
                .status
                                 (status
                                              ),
26
                .m_data
                                 (m_data
                                              ),
27
                .rf_data
                                 (rf_data
                                              ),
28
                .run
                                 (run
                                              ),
29
                .m_rf_addr
                                 (m_rf_addr ),
30
                .led
                                 (led
                                              ),
31
                .an
                                 (an
                                              ),
32
                                              )
                .seg
                                 (seg
33
           );
34
35
           cpu cpu(
36
                .clk
                                 (c1k
                                              ),
37
                                              ),
                .rst
                                 (rst
               .run
38
                                 (run
                                              ),
39
                .m_rf_addr
                                 (m_rf_addr ),
40
                .status
                                 (status
                                              ),
41
                .m_data
                                 (m_data
                                              ),
42
                .rf_data
                                 (rf_data
                                              )
43
           );
44
45
       endmodule
```

2.4.2 DBU模块具体实现

```
module Debug_Unit(
 2
        input clk,rst, //时钟, 复位
 3
        input succ, step,  //连续执行,单步执行
4
        input m_rf,inc,dec, //M/R选择,addr加减
 5
        input [2:0] sel,
                          //cpuc查看选择
 6
        input [6*32+16-1:0] status,
                                     //PC, IR, MD, A, B, ALUOut, signal
 7
        input [31:0] m_data,rf_data,
8
        output run, //cpu控制,数码管控制
9
        output [7:0] an,
10
        output reg [7:0] m_rf_addr,
11
        output reg [15:0] led,
12
        output [7:0] seg
13
        );
14
15
        //wire step_clr,inc_clr,dec_clr;
16
        wire step_p,inc_p,dec_p;
17
        reg [31:0] data;
        //信号处理,按键需处理,扳动无需处理
18
19
        //jitter_clr step_BTNC(clk,step,step_clr);
20
        //jitter_clr inc_BTNU(clk,inc,inc_clr);
        //jitter_clr dec_BUND(clk,dec,dec_clr);
21
22
        signal_edge step_edge(clk,step,step_p); //上板时记得修改
23
        signal_edge inc_edge(clk,inc,inc_p);
24
        signal_edge dec_edge(clk,dec,dec_p);
```

```
25
       assign run = succ|step_p; //run
26
27
       //m_rf_addr
28
29
       wire inc_dec;
30
       assign inc_dec = inc_p|dec_p;
31
       always @(posedge inc_dec,posedge rst)
32
       begin
33
           if(rst)
34
               m_rf_addr = 0;
35
           else
36
           begin
37
               case({inc_p,dec_p})
38
               2'b00,2'b11:m_rf_addr = m_rf_addr;
39
               2'b01:m_rf_addr = m_rf_addr-1;
               2'b10:m_rf_addr = m_rf_addr+1;
40
41
               endcase
42
           end
43
       end
44
45
46
       always @(*)
47
       begin
           led = \{\{4\{1'b0\}\}, status[11:0]\};
48
49
           case(sel)
           3'b000:
50
51
           begin
52
               led = \{\{8\{1'b0\}\}\}, m_rf_addr\};
53
               if(m_rf)
54
                   data = m_data;
55
               else
56
                   data = rf_data;
57
           3'b001:data = status[207:176]; //PC
58
           3'b010:data = status[175:144]; //IR
59
60
           3'b011:data = status[143:112]; //MD
61
           3'b100:data = status[111:80]; //A
62
           3'b101:data = status[79 :48]; //B
           63
64
           65
           endcase
66
       end
67
68
        seven_segment segment(clk,data,8'hff,an,seg);
69
70
    endmodule
```

• CPU模块相应修改(只展示修改部分)

```
1
   module cpu(
2
      input clk,rst,
3
      //为dbu增加端口
4
       input run,
5
       input [7:0] m_rf_addr, //dbu读地址
       output [31:0] m_data,rf_data,
6
                                    //R/D读出数据
7
       output [207:0] status //CPU内部状态
8
       );
```

```
9
10
       /*...*/
       //以下代码供dbu使用
11
       //修改时钟信号,后续时钟信号都用clk_p
12
13
       wire clk_p;
14
      assign clk_p = clk&run;
15
       assign status = {pc_cur,{IR_op,IR_rs,IR_rt,IR_Imm},MDR_out,
16
        regA, regB, ALUout, PCSource, PCwe, IorD, MemWrite, IRWrite, RegDst,
17
        MemtoReg,RegWrite,ALUm,ALUSrcA,ALUSrcB,ALUzero);
18
        /*...*/
19
20 endmodule
```

2.5 FPGA开发板测试:

返校后进行

三、实验总结

- 分析CPU的数据通路,并对其结构化描述,深入理解计算机硬件的基本组成、结构和工作原理;
- 调试器DBU的设计,增强了数字系统设计能力和调试数字系统的能力;
- 熟练掌握了数据通路和控制器的设计和描述方法。