《计算机组成原理》 实验报告



实验题目:<u>单周期 CPU</u>

学生姓名: 王志强

学生学号: PB18051049

完成日期: 2020.05.12

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. 单周期CPU

待设计的单周期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)
----------	------------	------------	------------	---------------	---------------

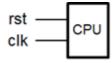
- |类指令
 - addi: rt <- rs+imm op = 001000
 lw: rt <- M(rs+addr) op = 100011
 sw: M(rs+addr) <- rt op = 101011
 - \circ beq: if(rs == rt) then pc <-pc+4+addr<<2; else pc <- pc+4 op = 000100

ſ	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)
```

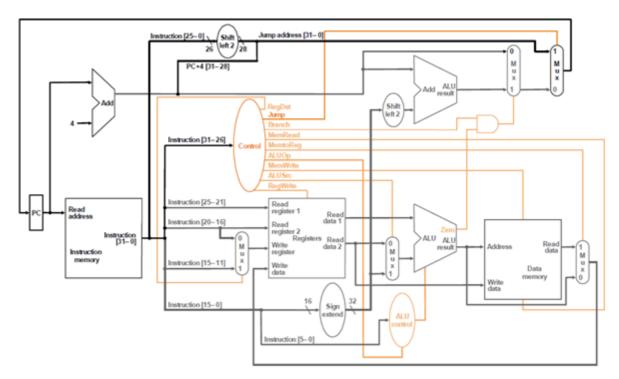
待设计的CPU逻辑符号和端口声明如下:



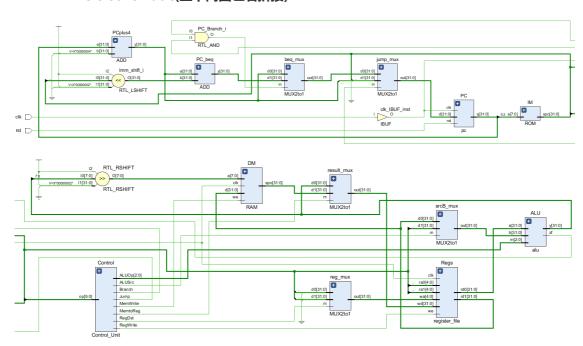
```
1 module cpu_one_cycle( //单周期CPU
2 input clk, //时钟(上升沿有效)
3 input rst //异步复位,高电平有效
4 );
5 ......
6 endmodule
```

分析以上待实现指令的功能,设计CPU的数据通路和控制单元(橙色部分)如图-2所示,其中ALU和寄存器堆可以利用实验1和实验2设计的模块来实现,指令存储器ROM和数据存储器RAM均采用IP例化实现,容量为256 x 32位的分布式存储器。

注意: ALU control被并入到了Control模块, ALUOp为3位宽, 直接传至ALU, 移位和符号拓展模块直接在顶层模块中进行行为描述



• RTL ANALYSIS-Schematic(上下两图左右拼接)



• 根据端口和功能要求,单周期cpu的顶层模块具体实现如下:

```
1
   module cpu(
 2
       input clk,rst
 3
       );
 4
 5
       wire [31:0] pc_cur,pc_next,pc_plus4,pc_jump,pc_beq,pc_result;
6
       wire [31:0] instr; //当前指令
 7
       wire [4:0] wa; //寄存器堆写入地址
       wire [31:0] Imm_signext,Imm_shift; //符号拓展后的立即数,左移后的立即数
8
       wire [31:0] reg_B; //寄存器堆端口2读出数据
9
10
       wire [31:0] srcA, srcB, ALU_result; //ALU操作数, 结果, SrcA也寄存器堆端口1
   数据
       wire [31:0] mem_dout, result; //DM输出,写回结果
11
```

```
12
        wire RegDst,Jump,Branch,MemtoReg,MemRead,MemWrite,ALUSrc,RegWrite;
    //控制信号
13
         wire [2:0] ALUOp;
14
         wire PC_Branch, zero;
15
16
17
         //data path
18
         pc PC(
19
             .clk
                              (c1k
                                                ),
20
             .rst
                              (rst
                                                ),
21
             .pc_in
                              (pc_next
                                                ),
22
                                                )
             .pc_out
                              (pc_cur
23
         ); //PC
24
         ADD PCplus4(
25
             .a
                              (pc_cur
                                                ),
26
             .b
                              (32'd4
                                                ),
27
                              (pc_plus4
                                                )
             . у
        ); //加法器实现pc+4
28
29
          assign pc_jump = {pc_plus4[31:28],instr[25:0],2'b00}; //j指令
30
31
         ROM IM(
32
                              (pc_cur[9:2]
                                                ),
             .a
33
                              (instr
                                                )
             .spo
34
         ); //instr memory
35
         MUX2to1 reg_mux(
36
             . m
                              (RegDst
                                                ),
37
             . d0
                              (instr[20:16]
                                                ),
38
             .d1
                              (instr[15:11]
                                                ),
39
             .out
                                                )
                              (wa
40
         ); //write rt or rd
41
42
         register_file Regs(
43
             .clk
                              (c1k
                                                ),
                                                ),
44
             .ra0
                              (instr[25:21]
                                                ),
45
             .rd0
                              (srcA
46
             .ra1
                              (instr[20:16]
                                                ),
47
             .rd1
                              (reg_B
                                                ),
48
             .we
                              (RegWrite
                                                ),
49
                              (wa
                                                ),
             .wa
             .wd
50
                              (result
                                                )
51
         ); //register file
52
         Control_Unit Control(
53
54
             .op
                              (instr[31:26]
                                                ),
                              (RegDst
55
             .RegDst
                                                ),
56
             .Jump
                              (Jump
                                                ),
57
             .Branch
                              (Branch
                                                ),
58
             .MemtoReg
                              (MemtoReg
                                                ),
                                                    //未使用
59
             .MemRead
                              (MemRead
                                                ),
                              (MemWrite
60
             .MemWrite
                                                ),
61
             .ALUSrc
                              (ALUSrc
                                                ),
             .RegWrite
62
                              (RegWrite
                                                ),
63
             .ALUOp
                              (ALUOp
                                                )
64
         ); //control unit
65
66
         MUX2to1 srcB_mux(
67
             . m
                              (ALUSrc
                                                ),
68
             .d0
                              (reg_B
                                                ),
```

```
69
              .d1
                               (Imm_signext
                                                ),
 70
              .out
                               (srcB
                                                )
                 //src
 71
          );
 72
          alu ALU(
 73
              .a
                               (srcA
                                                ),
 74
              .b
                               (srcB
                                                ),
 75
                               (ALUOp
                                                ),
              . m
 76
                               (ALU_result
              . у
                                                ),
 77
              .zf
                               (zero
                                                )
 78
          ); //alu
 79
          assign Imm_signext = {{16{instr[15]}},instr[15:0]}; //sign extend
 80
 81
          assign Imm_shift = (Imm_signext<<2);  //shift left2</pre>
 82
 83
          ADD PC_beq(
 84
                               (pc_plus4
                                                ),
              .a
 85
              .b
                               (Imm_shift
                                                ),
 86
                               (pc_beq
                                                )
              . У
 87
          ); //pc_beq or pc_plus4
 88
          assign PC_Branch = zero&Branch;
                                                //PC_Branch
 89
          MUX2to1 beq_mux(
 90
                               (PC_Branch
                                                ),
              . m
 91
              .d0
                               (pc_plus4
                                                ),
 92
              .d1
                               (pc_beq
                                                ),
 93
              .out
                               (pc_result
 94
          );
 95
          MUX2to1 jump_mux(
 96
              . m
                                                ),
                               (Jump
 97
              .d0
                               (pc_result
                                                ),
                               (pc_jump
 98
              .d1
                                                ),
 99
              .out
                               (pc_next
                                                )
100
          );
101
102
          RAM DM(
103
              .clk
                               (c1k
                                                ),
104
              .we
                               (MemWrite
                                                ),
105
                               (ALU_result[9:2]), //特别注意指令按字节寻址
              .a
              .d
106
                               (reg_B
                                                ),
107
                               (mem_dout
                                                )
              .spo
108
          );
109
          MUX2to1 result_mux(
110
              . m
                               (MemtoReg
                                                ),
111
              .d0
                               (ALU_result
                                                ),
112
              .d1
                               (mem_dout
                                                ),
113
              .out
                               (result
                                                )
114
         );
115
116
     endmodule
```

• 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 );
```

```
//实现以下6条指令
6
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;
12
        parameter j
                     = 6'b000010;
        //add_op = 000, sub_op = 000
13
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;
24
            sw : control = 11'bx00x0110000;
25
            beg : control = 11'bx01x0000001;
            j : control = 11'bx1xx00x0xxx;
26
            default: control = 11'bxxxxxxxxxx;
27
28
            endcase
29
        end
    endmodule
```

• 仿真MIPS指令

```
# 本文档存储器以字节编址
1
   # 初始PC = 0x00000000
 2
 3
4
   .data
       .word 0,6,0,8,0x80000000,0x80000100,0x100,5,0 #编译成机器码时,编译器
 5
   会在前面多加个0, 所以后面1w指令地址会多加4
6
7
   _start:
8
           addi $t0,$0,3
                                #t0=3
           addi $t1,$0,5
9
                                #t1=5
                                       4
10
           addi $t2,$0,1
                                #t2=1
                                        8
11
           addi $t3,$0,0
                                #t3=0 12
12
13
           add $s0,$t1,$t0
                                #s0=t1+t0=8 测试add指令
                                                        16
14
           1w
               $s1,12($0)
                                                         20
15
           beq $s1,$s0,_next1
                               #正确跳到_next
                                                        24
16
17
           j _fail
18
19
   _next1:
20
           lw $t0, 16($0)
                                #t0 = 0x80000000
                                                32
21
           lw $t1, 20($0)
                                #t1 = 0x80000100
                                                36
22
           add $s0,$t1,$t0
                                #s0 = 0x00000100 = 256 40
23
24
           lw $s1, 24($0)
                                                      44
25
           beq $s1,$s0,_next2 #正确跳到_success
                                                     48
26
27
           j_fail
28
```

```
29 _next2:
         add $0, $0, $t2 #$0应该一直为0
beq $0,$t3,_success #
30
                                               56
31
                                                60
32
33 _fail:
        sw $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
34
   址0x08里值为0
         j _fail
35
36
37
   _success:
        sw $t2,8($0) #全部测试通过,存储器地址0x08里值为1
38
39
         j _success
                     #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
40
   过测试
```

• cpu仿真波形及解释

此处可以看到7条指令的仿真(4条addi, 1条add, 1条lw, 1条beq), 图中可以清楚看到指令执行时各部件的端口值,如pc值、寄存器堆输出端口、ALU结果、控制信号、data memory输出等。 pc_cur = 24时,为beq指令,可以看到分支成功,pc_next = 32,转而执行lw指令

Name	Value	0 ns	5 ns	10 ns	15 ns	20 ns	25 ns	30 ns	35 ns	40 ns	45 ns	50 ns	55 ns	60 ns	65 ns	70 ns
₩ clk	1															
¼ rst	0															
> W pc_cur[31:0]	32		0			4	χ	3		12	1	16	:	20	2	24
> ⊌ pc_next[31:0]	36		4			8	1	2		16	2	20		24	з	32
> W instr[31:0]	8c080010		20080003	,	2009	90005	200ε	.0001	2001	0000	0128	8020	8c1	1000c	1230	00001
> W wa[4:0]	8		8			9	1	0	1	1	1	6		17		X
> W reg_B[31:0]	3	0		3)				3		0		8
> W srcA[31:0]	0					0						5		0		8
> W srcB[31:0]	16		3			5	X			0		3		12		8
> WALU_result[31:0]	16		3			5	X			0		8		12		0
> * mem_dout[31:0]	-2147483648		0			6	X			0				8		0
> * result[31:0]	-2147483648		3			5	X			0			8			0
1⊌ RegDst	0															
1⊌ Jump	0															
1⊌ Branch	0															
1 MemtoReg	1															
1⊌ MemWrite	0															
1 ALUSrc	1															
1⊌ RegWrite	1															
> W ALUOp[2:0]	0							0								1
1⊌ PC_Branch	0															
1⊌ zero	0															

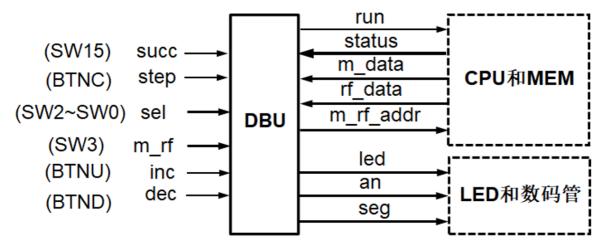
下图仿真了剩余指令,其中包括beq,add,lw,sw,j指令。

pc_cur = 60时,为j指令,可以看到pc_next = 72,分支成功,最后根据mem_dout = 1可知x08 = 1

																					1	25-000
Name	Value	75	ns	80 ns	85 ns	90 ns	95 ns	100 ns	105 ns	110 ns	115 ns	120 ns	125 ns	130 ns	135 ns	140 ns	145 ns	150 ns	155 ns	160 ns	165 ns	170 ns
₩ clk	1																					
14 rst	0																					
> № pc_cur[31:0]	76			32		36		40		44		18		56	(50		72		76) ·	72
▶ № pc_next[31:0]	72			36		40		44		48		56	,	60		2		76		72)	76
₩ instr[31:0]	08000012		8c0	30010	86	090014	012	88020	8c1	10018	123	00001	000:	10020	1001	0002	ac0a	10008	080	00012	ac0:	a0008
₩ wa[4:0]	0			8		9		16		17		X		Ó		X		X		0		х
▼reg_B[31:0]	0)		3		5	-2147	483648	X	8	2	56		1		0		1		0	1	1
₩ srcA[31:0]	0	χ_			0		-2147	483392	X	0	2	56						0				
⊌ srcB[31:0]	X	X	16 20		-2147483648		24		256		!		•		8		х		8			
■ ALU_result[31:0]	0	<u> </u>	16 20		256		24		0		1		<u> </u>	· ·		8		0 8		8		
■ mem_dout[31:0]	0	1	-2147	483648	-21	47483392		0	2	256			,			0						1
result[31:0]	0	1	-2147	483648	-21	47483392		. 2	56			0		1		0		X		0		
1⊌ RegDst	X									_												
1 Jump	1						<u> </u>	_													_	
I Branch	X																				0	
MemtoReg	X																					
1 MemWrite	0																					
1 ALUSrc	X																					
1⊌ RegWrite	0																					
₩ALUOp[2:0]	X						0					1		0		1		0		X		0
1 PC_Branch	X					—																
a zero	1																					

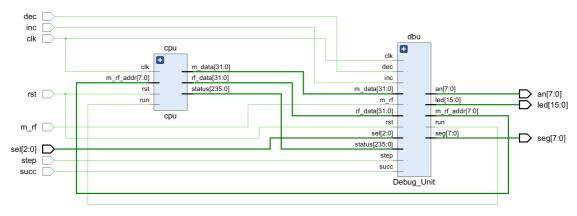
2、调试单元Debug Unit(DBU)

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。

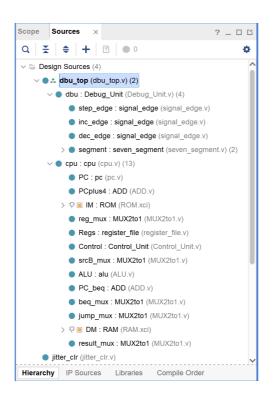


注:图中省略了clk和rst信号

• RTL ANALYSIS-Schematic(顶层模块——连接图)



• 文件架构



• 顶层模块具体实现

```
module dbu_top(
 1
 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
 6
        output [15:0] led, //led
 7
        output [7:0] an, //seven segment enable
 8
        output [7:0] seg //seven segment output
9
        );
10
11
        wire run;
12
        wire [235:0] status;
13
        wire [31:0] m_data,rf_data;
        wire [7:0] m_rf_addr;
14
        //data path
15
16
        Debug_Unit dbu(
17
            .clk
                            (clk
                                        ),
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
            .run
28
                            (run
                                        ),
29
            .m_rf_addr
                            (m_rf_addr ),
30
            .led
                                        ),
                            (led
31
            .an
                            (an
                                        ),
32
                                        )
            .seg
                            (seg
33
        );
34
35
        cpu cpu(
            .clk
                            (c1k
36
                                        ),
37
            .rst
                            (rst
                                        ),
38
            .run
                            (run
                                        ),
39
            .m_rf_addr
                            (m_rf_addr ),
40
            .status
                            (status
                                        ),
41
            .m_data
                            (m_data
                                        ),
            .rf_data
                            (rf_data
42
                                        )
43
        );
44
    endmodule
```

• 核心模块DBU具体实现

```
1 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 [235:0] status,
//pc_in,pc_out,instr,rf_rd1,rf_rd2,alu_y,m_rd,signal
```

```
input [31:0] m_data,rf_data,
                                       //数据读出
 8
        output run, //cpu控制,数码管控制
9
        output [7:0] an,
                            //数码管使能
10
        output reg [7:0] m_rf_addr, //地址
11
        output reg [15:0] led, //led
        output [7:0] seg //数码管显示信号
12
13
        );
14
        //wire step_clr,inc_clr,dec_clr;
15
16
        wire step_p,inc_p,dec_p;
17
        reg [31:0] data;
18
        //信号处理,按键需处理,扳动无需处理
19
        //上板时要去抖动, 仿真时先注释
        //jitter_clr step_BTNC(clk,step,step_clr);
20
21
        //jitter_clr inc_BTNU(clk,inc,inc_clr);
22
        //jitter_clr dec_BUND(clk,dec,dec_clr);
23
        signal_edge step_edge(clk,step,step_p);
24
        signal_edge inc_edge(clk,inc,inc_p);
25
        signal_edge dec_edge(clk,dec,dec_p);
26
27
        assign run = succ|step_p; //run信号
28
        //m_rf_addr
29
        wire inc_dec;
30
                                       //fpga触发器只能有一个复位、一个时钟
31
        assign inc_dec = inc_p|dec_p;
        always @(posedge inc_dec,posedge rst)
32
33
        begin
            if(rst)
34
35
                m_rf_addr = 0;
36
            else
37
            begin
38
                case({inc_p,dec_p})
                2'b00,2'b11:m_rf_addr <= m_rf_addr;
39
40
                2'b01:m_rf_addr <= m_rf_addr-1;
41
                2'b10:m_rf_addr <= m_rf_addr+1;
42
                endcase
43
            end
44
        end
45
46
47
        always @(*)
48
        begin
49
            led = {{4{1'b0}}},status[11:0]};
50
            case(sel)
51
            3'b000:
52
            begin
53
                led = \{\{8\{1'b0\}\}, m_rf_addr\};
54
                if(m_rf)
55
                    data = m_data;
56
                else
57
                    data = rf_data;
58
59
            3'b001:data = status[235:204];
                                             //pc_in
60
            3'b010:data = status[203:172];
                                             //pc_out
61
            3'b011:data = status[171:140];
                                             //instr
62
            3'b100:data = status[139:108];
                                             //rf_rd1
63
            3'b101:data = status[107: 76];
                                             //rf_rd2
64
            3'b110:data = status[75 : 44];
                                             //alu_y
```

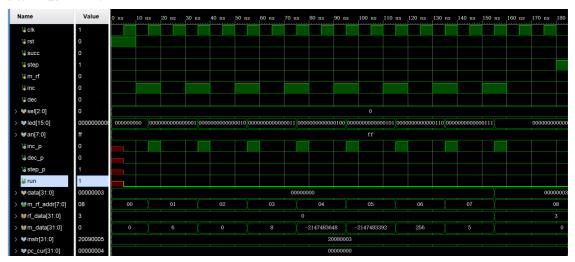
```
3'b111:data = status[43 : 12]; //m_rd
endcase
end
seven_segment segment(clk,data,8'hff,an,seg);
endmodule
endmodule
```

• CPU模块相应修改(只展示修改部分)

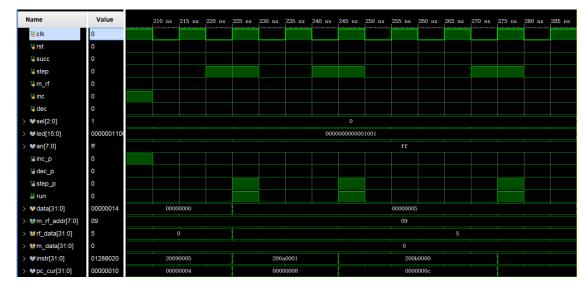
```
module cpu(
 1
 2
        input clk, rst,
 3
        //为dbu增加端口
 4
       input run,
        input [7:0] m_rf_addr, //dbu读地址
 5
 6
       output [31:0] m_data, rf_data, //R/D读出数据
 7
       output [235:0] status //CPU内部状态
 8
       );
9
       /*...*/
10
       //以下代码供dbu使用
11
12
        //修改时钟信号,后续时钟信号都用clk_p
13
       wire clk_p;
        assign clk_p = clk&run;
14
15
        assign status =
    {pc_next,pc_cur,instr,srcA,reg_B,ALU_result,mem_dout,
16
     Jump,Branch,RegDst,RegWrite,MemRead,MemtoReg,MemWrite,ALUOp,ALUSrc,zer
    o};
17
        /*...*/
18
19
    endmodule
```

• DBU波形仿真及解释

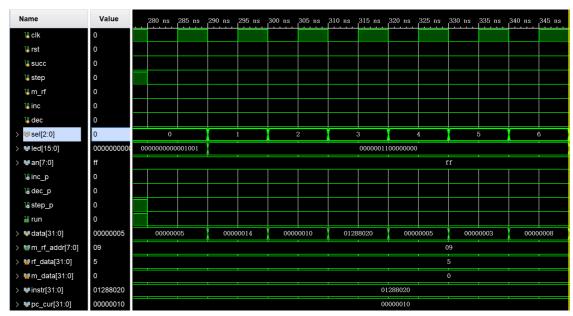
这里显示了我们的第一条指令,图中可以看到sel、step、inc的控制效果以及各端口的输出值。可以看到,当m_rf_addr=8且m_rf=0时,输出寄存器t0的值,为3,与指令结果相符,同时led显示我们的输入地址值8:



下面,我们可以看到后续若干条指令的执行,以及Register File和Data Memory的输出:



接下来,在ADD指令下,我们切换不同的sel信号,来验证输出的正确性:



检察后续若干条指令以及sel=7的输出值:



翻到最后,可以看到m_data=1,符合预期:



- 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。

四、思考题

Q: 修改数据通路和控制器,增加支持如下指令:

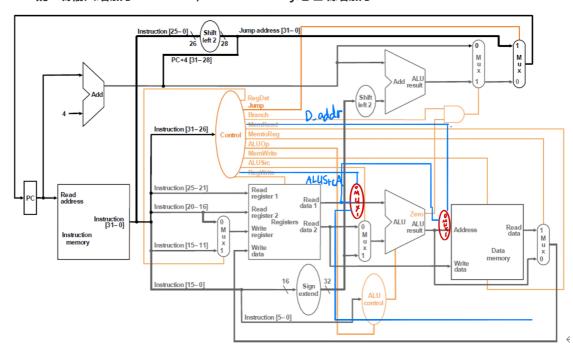
accm: rd <- M(rs) + rt; op = 000000, funct = 101000

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

A: 设计方法、控制器代码如下:

• 数据通路:

增加了控制信号SrcA、D_addr分布作为ALU的A端输入选择和Data Memory地址输入选择,在ALU的A端输入增加了MUX2to1,Data Memory地址端增加了MUX2to1



• Control Unit:

```
1
    module Control_Unit(
 2
        input [5:0] op,
 3
        output
    RegDst,Jump,Branch,MemtoReg,MemRead,MemWrite,ALUSrc,RegWrite,
        output [2:0] ALUOp, //待实现的指令有限,故直接生成alu控制信号
4
 5
        output SrcA,D_addr
 6
        );
 7
        parameter add = 6'b0000000;
8
        parameter addi = 6'b001000;
9
        parameter lw = 6'b100011;
10
        parameter sw = 6'b101011;
        parameter beq = 6'b000100;
11
12
        parameter j
                       = 6'b000010;
        parameter accm = 6'b0000000;
13
        //add_op = 000, sub_op = 000
14
15
        reg [12:0] control;
16
        assign {RegDst, Jump, Branch, MemtoReg, MemRead, MemWrite,
17
                ALUSrc, RegWrite, ALUOp, SrcA, D_addr}=control;
18
19
        always @(op)
20
        begin
21
            case(op)
22
            add : control = 13'b1000000100000;
23
            addi: control = 13'b0000001100000;
            lw : control = 13'b0001101100000;
24
25
            sw : control = 13'bx00x011000000;
26
            beq : control = 13'bx01x000000100;
27
            j : control = 13'bx1xx00x0xxx00;
28
            accm: control = 13'b1000100100011;
29
            default: control = 13'bxxxxxxxxxxx;
30
            endcase
31
        end
32 endmodule
```