中国科学院大学计算机组成原理实验课

实 验 报 告

学号: 2015K8009929045 姓名: 张远航 专业: 计算机科学与技术

实验序号: 3 实验名称: 多周期处理器设计

注 1: 本实验报告请以 PDF 格式提交。文件命名规则: [学号]-PRJ[实验序号]-RPT.pdf，其中文件名字母大写，后缀名小写。例如: [2014K8009959088]-PRJ[1]-RPT.pdf
注 2: 实验报告模板以下部分的内容供参考，可包含但不限定如下条目内容。

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等）

数据通路模块代码如下，实现遵照 Patterson 在 COD 中给出的结构，为了实现 LUI、SLL 等新操作加宽了 ALUSrc、RegDst 等信号。电路结构参见图 1 。

```
module datapath(
    ... (port definition, omitted)
);

    // additional registers
    reg  [31:0] MDR, IR, ALUOut, A, B;

    wire [4:0] rs, rt, rd, sa;
    wire [15:0] immediate;

    reg  [31:0] nPC;

    wire [31:0] sext_imm;

    initial begin
        nPC = 32'b0;
    end

    assign Op = IR[31:26];
    assign rs = IR[25:21];
    assign rt = IR[20:16];
```

```
22      assign rd = IR[15:11];
23      assign sa = IR[10:6];
24
25      assign immediate = IR[15:0];
26      assign sext_imm = {{(16){IR[15]}}, immediate};
27      // we assign the Op field to func for I-type instructions
28      assign func = (ALUOp == 2'b11) ? IR[31:26]: IR[5:0];
29
30      // program counter
31      always @(*)
32          case (PCSource)
33              2'b00: nPC <= Result;
34              2'b01: nPC <= ALUOut;
35              2'b10: nPC <= {mips_PC[31:28], IR[25:0], 2'b00};
36              default: nPC <= Result;
37          endcase
38
39      // ALU
40      always @(*)
41          case (ALUSrcA)
42              2'b00: OpA = mips_PC;
43              2'b01: OpA = A;
44              2'b10: OpA = sa;
45              2'b11: OpA = 32'd16;
46          endcase
47
48      always @(*)
49          case (ALUSrcB)
50              2'b00: OpB = B;
51              2'b01: OpB = 32'd4;
52              2'b10: OpB = sext_imm;
53              2'b11: OpB = sext_imm << 2;
54          endcase
55
56      // register file
57      always @(*)
58          case (RegDst)
59              2'b00: waddr = rt;
60              2'b01: waddr = rd;
```

```
61              2'b10: waddr = 5'd31;   // JAL
62              default: waddr = 5'd0;
63          endcase
64
65      assign raddr1 = rs;
66      assign raddr2 = rt;
67      assign wen = RegWrite;
68      always @(*)
69          case (MemtoReg)
70              2'b00: wdata = ALUOut;
71              2'b01: wdata = MDR;
72              2'b10: wdata = Result;
73              default: wdata = Result;
74          endcase
75
76      // MIPS
77      assign mips_MemWrite = MemWrite;
78      assign mips_MemRead = MemRead;
79      assign mips_Address = ALUOut;
80      assign mips_Write_data = B;
81
82      always @(posedge clk) begin
83          if (rst) begin
84              IR <= 32'b0; MDR <= 32'b0;
85              mips_PC <= 32'b0;
86              A <= 32'b0; B <= 32'b0;
87              ALUOut <= 32'b0;
88          end
89          else begin
90              if (PCwen) mips_PC <= nPC;
91              if (IRWrite) IR <= mips_Instruction;
92              MDR <= mips_Read_data;
93              A <= rdata1;
94              B <= rdata2;
95              ALUOut <= Result;
96          end
97      end
98
99  endmodule
```

采用多级译码的 ALU 的控制单元，结构见图 2（为了区分不同的 SPECIAL 和 I 型指令，输入需增加 func 信号）：

```
module alu_control(
    input  [5:0] func,

    input  [1:0] ALUOp,
    output reg [2:0] ALUctr
);

    ... (func code parameters, ommited)
    always @(*) begin
        case (ALUOp)
            2'b00: ALUctr <= OP_ADD;    // LW or SW
            2'b01: ALUctr <= OP_SUB;    // BEQ or BNE
            2'b10:                      // R-type
                case (func)
                    SLL:  ALUctr <= OP_SLL;
                    ADDU: ALUctr <= OP_ADD;
                    OR:   ALUctr <= OP_OR;
                    // [$zero]+GPR[rs] = GPR[rs]
                    JR:   ALUctr <= OP_ADD;
                    SLT:  ALUctr <= OP_SLT;
                    default: ALUctr <= OP_NOP;
                endcase
            2'b11:                      // I-type or J-type
                case (func)
                    ADDIU: ALUctr <= OP_ADD;
                    SLTI:  ALUctr <= OP_SLT;
                    SLTIU: ALUctr <= OP_SLTU;
                    LUI:   ALUctr <= OP_LUI;
                    JAL:   ALUctr <= OP_ADD;
                    default: ALUctr <= OP_NOP;
                endcase
        endcase
    end

endmodule
```

以及 CPU 的控制单元（多周期加入了 FSM）：

```
1  module control(
2      input  clk, rst, Zero,
3
4      input  [5:0] Op, func,
5      output reg [1:0] ALUOp, ALUSrcB, ALUSrcA, PCSource, RegDst,
           MemtoReg,
6
7      output reg PCWriteCond, PCWrite, MemRead, MemWrite, IRWrite,
            RegWrite,
8      output wire PCwen
9  );
10
11     ... (opcode definition, omitted)
12     // FSM data selector
13     parameter FETCH = 4'b0000;          // instruction fetch
14     parameter DECODE = 4'b0001;         // instruction decode /
           register fetch
15     parameter MEMADDRCOMP = 4'b0010;    // memory address
           computation
16     parameter MEMACCESS_L = 4'b0011;    // memory access (LW)
17     parameter MEMRDEND = 4'b0100;       // memory read
           completion step
18     parameter MEMACCESS_S = 4'b0101;    // memory access (SW)
19     parameter RTYPEEXEC = 4'b0110;      // R-type execution
20     parameter RTYPEEND = 4'b0111;       // R-type completion
21     parameter BRANCHEND = 4'b1000;      // branch completion
22     parameter JMPEND = 4'b1001;         // jump completion
23     parameter ITYPEEXEC = 4'b1010;      // I-type execution
24     parameter ITYPEEND = 4'b1011;       // I-type completion
25     parameter JALEXEC = 4'b1100;        // JAL execution, step 1
26     parameter JREXEC = 4'b1101;         // JR execution
27     parameter SLLEXEC = 4'b1110;
28     parameter LUIEXEC = 4'b1111;
29
30     reg [3:0] state, nextstate;
31
32     // BNE or BEQ
33     assign PCwen = PCWrite | (PCWriteCond & (Op[0] ? !Zero :
           Zero));
```

```
34
35      // synchronous reset
36      always @(posedge clk)
37          if (rst)
38              state <= FETCH;
39          else
40              state <= nextstate;
41
42      // next state logic
43      always @(state or Op or func) begin
44          case (state)
45              FETCH: nextstate <= DECODE;
46              DECODE: case (Op)
47                  SPECIAL:
48                      case (func)
49                          JR: nextstate <= JREXEC;
50                          SLL: nextstate <= SLLEXEC;
51                          default: nextstate <= RTYPEEXEC;
52                      endcase
53                  ADDIU, SLTI, SLTIU: nextstate <= ITYPEEXEC;
54                  LUI: nextstate <= LUIEXEC;
55                  BEQ, BNE: nextstate <= BRANCHEND;
56                  J: nextstate <= JMPEND;
57                  JAL: nextstate <= JALEXEC;
58                  LW: nextstate <= MEMADDRCOMP;
59                  SW: nextstate <= MEMADDRCOMP;
60                  default: nextstate <= FETCH;
61              endcase
62              MEMADDRCOMP: case(Op)
63                  LW: nextstate <= MEMACCESS_L;
64                  SW: nextstate <= MEMACCESS_S;
65                  default: nextstate <= FETCH;
66              endcase
67              JALEXEC: nextstate <= JMPEND;   // jump after PC+4
68              JREXEC: nextstate <= FETCH;
69              LUIEXEC: nextstate <= ITYPEEND;
70              SLLEXEC: nextstate <= RTYPEEND;
71              MEMACCESS_L: nextstate <= MEMRDEND;
72              RTYPEEXEC: nextstate <= RTYPEEND;
```

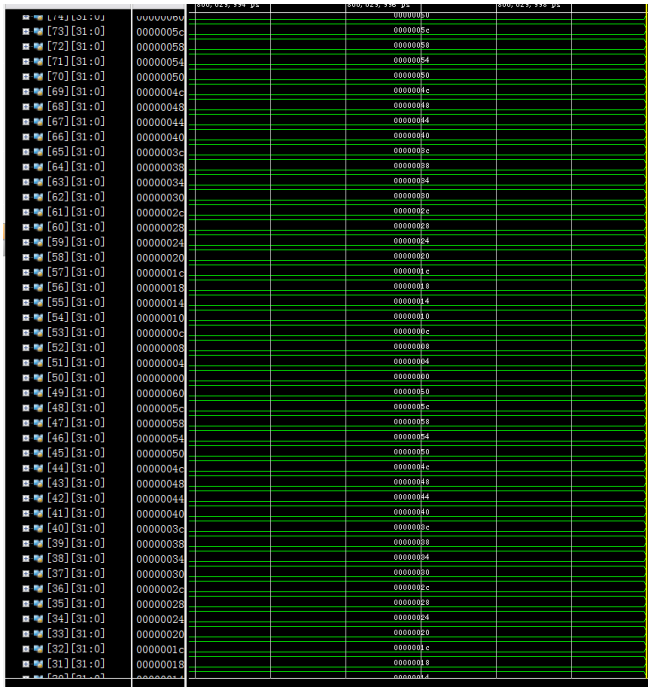
```
73              ITYPEEXEC: nextstate <= ITYPEEND;
74              MEMACCESS_S, MEMRDEND, ITYPEEND, RTYPEEND, BRANCHEND
                    , JMPEND: nextstate <= FETCH;
75              default: nextstate <= FETCH;
76          endcase
77      end
78
79      always @(state) begin
80          case (state)
81              FETCH:  {PCWriteCond, PCWrite, MemRead, MemWrite,
                    MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                    ALUSrcB, RegWrite, RegDst} <= 18'
                    b0110_00_1_00_00_00_01_0_00;
82              DECODE: {PCWriteCond, PCWrite, MemRead, MemWrite,
                    MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                    ALUSrcB, RegWrite, RegDst} <= 18'
                    b0000_00_0_00_00_00_11_0_00;
83              MEMADDRCOMP: {PCWriteCond, PCWrite, MemRead,
                    MemWrite, MemtoReg, IRWrite, PCSource, ALUOp,
                    ALUSrcA, ALUSrcB, RegWrite, RegDst} <= 18'
                    b0000_00_0_00_00_01_10_0_00;
84              MEMACCESS_L: {PCWriteCond, PCWrite, MemRead,
                    MemWrite, MemtoReg, IRWrite, PCSource, ALUOp,
                    ALUSrcA, ALUSrcB, RegWrite, RegDst} <= 18'
                    b0010_00_0_00_00_00_00_0_00;
85              MEMACCESS_S: {PCWriteCond, PCWrite, MemRead,
                    MemWrite, MemtoReg, IRWrite, PCSource, ALUOp,
                    ALUSrcA, ALUSrcB, RegWrite, RegDst} <= 18'
                    b0001_00_0_00_00_00_00_0_00;
86              RTYPEEXEC: {PCWriteCond, PCWrite, MemRead, MemWrite,
                     MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                    ALUSrcB, RegWrite, RegDst} <= 18'
                    b0000_00_0_00_10_01_00_0_00;
87              RTYPEEND:  {PCWriteCond, PCWrite, MemRead, MemWrite,
                     MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                    ALUSrcB, RegWrite, RegDst} <= 18'
                    b0000_00_0_00_00_00_00_1_01;
88              JREXEC:    {PCWriteCond, PCWrite, MemRead, MemWrite,
                     MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
```

```
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0100_00_0_00_10_01_00_0_00;
89          SLLEXEC:   {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_00_0_00_10_10_00_0_00;
90          LUIEXEC:   {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_00_0_00_11_11_10_0_00;
91          // for the time being we use PC+4 for JAL
92          JALEXEC:   {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_10_0_00_11_00_01_1_10;
93          ITYPEEXEC: {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_00_0_00_11_01_10_0_00;
94          ITYPEEND:  {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_00_0_00_00_00_00_1_00;
95          BRANCHEND: {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b1000_00_0_01_01_01_00_0_00;
96          MEMRDEND:  {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0000_01_0_00_00_00_00_1_00;
97          JMPEND:    {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'
                b0100_00_0_10_00_00_00_0_00;
98          default:   {PCWriteCond, PCWrite, MemRead, MemWrite,
                 MemtoReg, IRWrite, PCSource, ALUOp, ALUSrcA,
                ALUSrcB, RegWrite, RegDst} <= 18'b0;
99      endcase
```

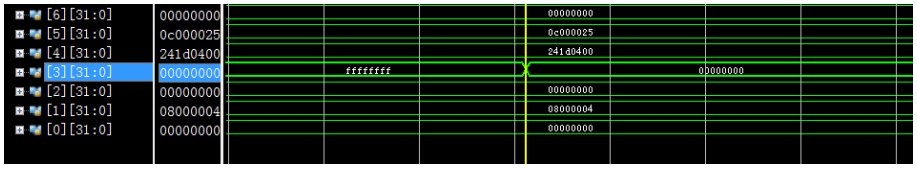
```
100        end
101
102    endmodule
```

结构见图 3 。

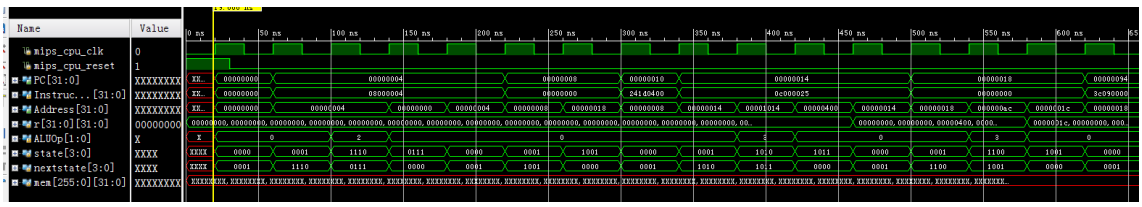
下面是一些仿真波形：



memcpy 程序执行结果



上板用程序执行结果，执行结束标志变量变为 0（以 bubble-sort 为例）



状态机状态信号的变化情况

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真及上板调试过程中的难点等）

Bug 合集

(a) 数据通路里，有信号在实例化的时候忘记接线了。

(b) 译码器又有写错的……

(c) 部分信号没有放进 always 块的敏感性列表里导致出错。

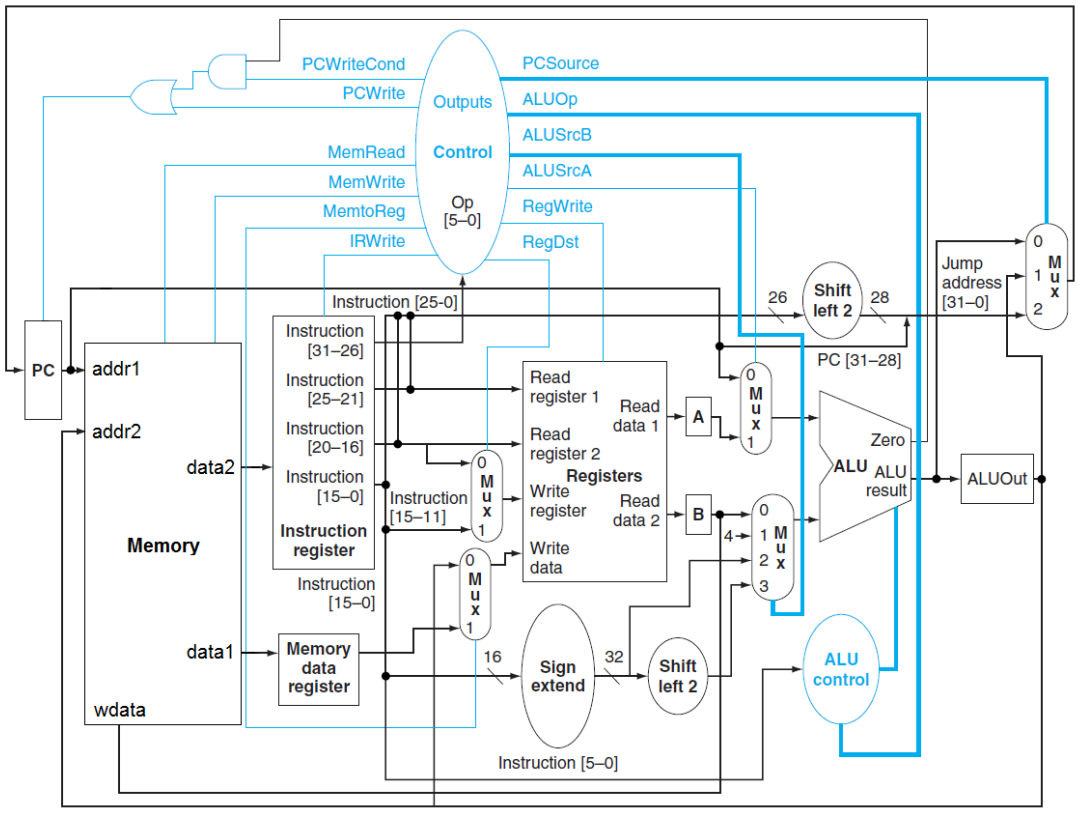
三、 对讲义中思考题的理解和回答

无思考题

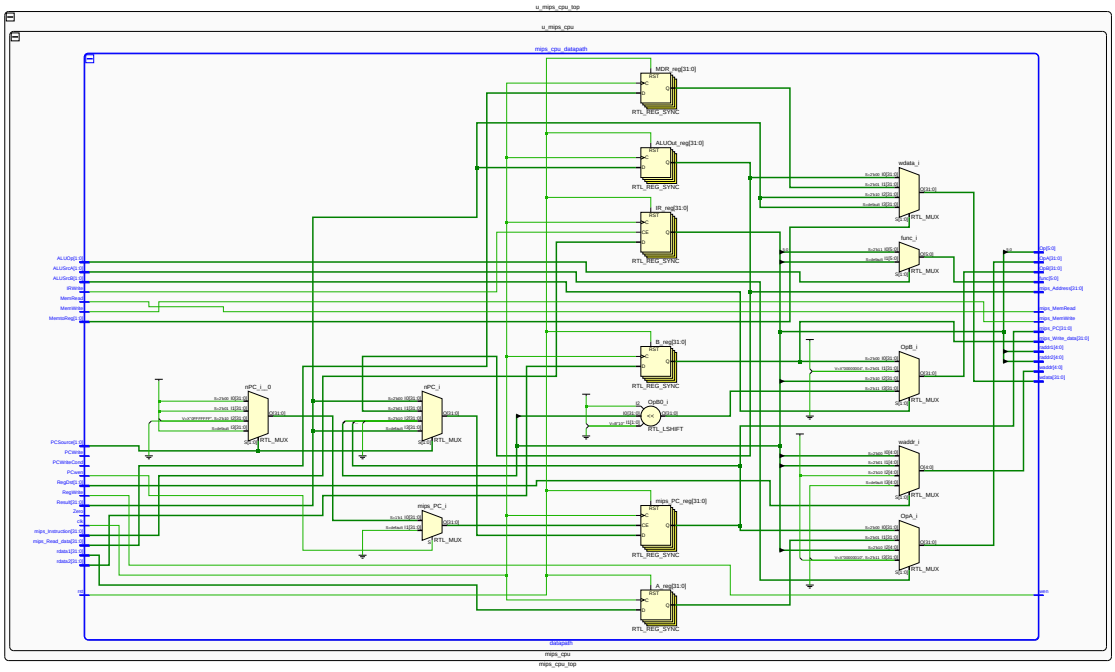
四、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，以及其他想与任课老师交流的内容等）

- 有了前两次实验的基础，这次实验终于开始感觉比较轻松了，所以没有太多想说的……
- 原来.vh 文件是可以直接包含进去的==
- 写译码器的时候中间插入下划线作为分隔符给了我很大的视觉辅助。
- 被助教哥哥的大规模测试用例吸引住了，自己也想搞一个了！

插图



(a) 期望的数据通路（来源：Patterson & Hennessey, 2005，有改动）



(b) 实现的 RTL 图

图 1: 数据通路

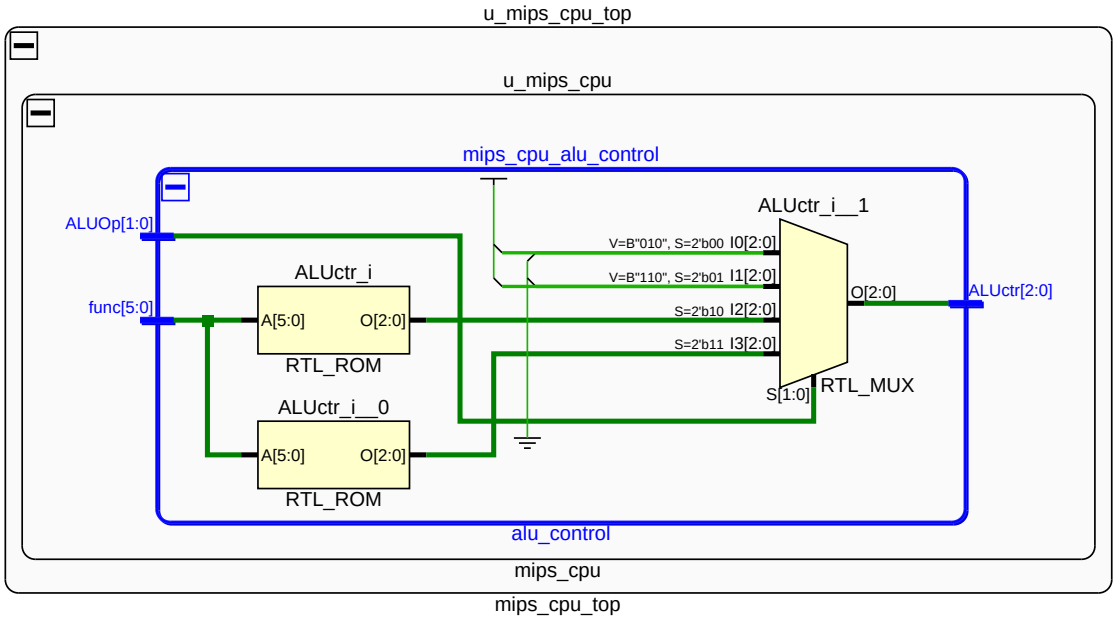

u_mips_cpu_top
u_mips_cpu
mips_cpu_alu_control
ALUOp[1:0]
func[5:0]
ALUctr_i
A[5:0]
O[2:0]
RTL_ROM
ALUctr_i__0
A[5:0]
O[2:0]
RTL_ROM
ALUctr_i__1
V=B"010", S=2'b00 I0[2:0]
V=B"110", S=2'b01 I1[2:0]
S=2'b10 I2[2:0]
S=2'b11 I3[2:0]
O[2:0]
S[1:0]
RTL_MUX
ALUctr[2:0]
alu_control
mips_cpu
mips_cpu_top

图 2: ALU 控制单元

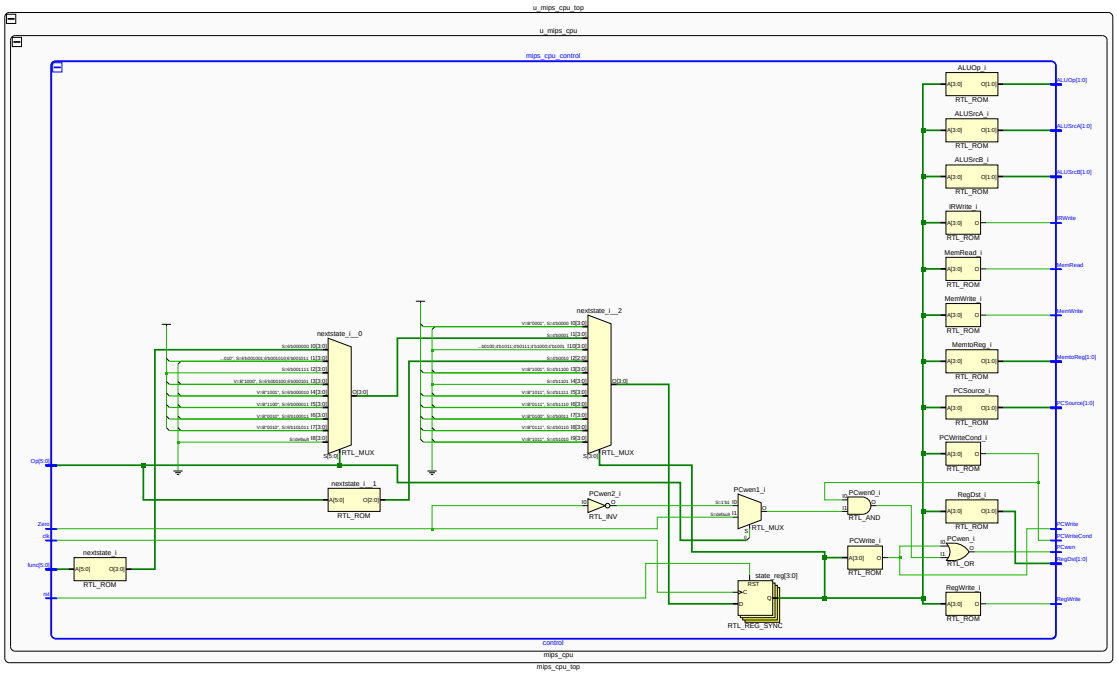

u_mips_cpu_top
u_mips_cpu
mips_cpu_control
nextstate_i__0
nextstate_i__2
RTL_MUX
nextstate_i__1
RTL_ROM
PCwen2_i
RTL_INV
PCwen1_i
RTL_MUX
PCwen0_i
RTL_AND
PCwen_i
RTL_OR
PCWrite_i
RTL_ROM
state_reg[3:0]
RTL_REG_SYNC
nextstate_i
ALUOp_i
ALUSrcA_i
ALUSrcB_i
IRWrite_i
MemRead_i
MemWrite_i
MemtoReg_i
PCSource_i
PCWriteCond_i
RegDst_i
RegWrite_i
control
mips_cpu
mips_cpu_top

图 3: 控制单元