实验 3：简单单周期 CPU

张子康 PB22020660

2024 年 04 月 15 日

1 实验目的与内容

1.1 实验目的

实现一个最为基础的单周期 CPU。

1.2 实验内容

1.2.1 译码器设计

根据自己选择的指令集，设计译码器 Decoder 模块，以正确生成控制信号。

1.2.2 搭建 CPU

正确实现 CPU 的各个功能模块，并根据数据通路将其正确连接。理论上，只需要完成 CPU 模块及其子模块的设计，而无需修改其他模块的内容。最终，你需要在 FPGAOL 上上板运行，并通过我们给出的测试程序，为此你需要实现 Lab1 列出的指令中，从 add.w（add）到 pcaddu12i（auipc）之间的全部指令。

2 逻辑设计

2.1 译码器设计

```
1 `define ADD                   5'B00000
2 `define SUB                   5'B00010
3 `define SLT                   5'B00100
4 `define SLTU                  5'B00101
5 `define AND                   5'B01001
6 `define OR                    5'B01010
7 `define XOR                   5'B01011
8 `define SLL                   5'B01110
9 `define SRL                   5'B01111
```

```
10 `define SRA                  5'B10000
11 `define SRC0                 5'B10001
12 `define SRC1                 5'B10010
13 module DECODE (input [31 : 0] inst,
14                output [4 : 0] alu_op,
15                output [31 : 0] imm,
16                output [4 : 0] rf_ra0,
17                output [4 : 0] rf_ra1,
18                output [4 : 0] rf_wa,
19                output [0 : 0] rf_we,
20                output [0 : 0] alu_src0_sel,
21                output [0 : 0] alu_src1_sel);
22 reg [4:0] op;
23 reg [0:0] we;
24 reg [4:0] ra0;
25 reg [4:0] ra1;
26 reg [4:0] wa;
27 reg [0:0] src0_sel;
28 reg [0:0] src1_sel;
29 reg [31:0] imm_;
30
31 //初始化寄存器
32 initial begin
33     op       = 0;
34     we       = 0;
35     ra0      = 0;
36     ra1      = 0;
37     wa       = 0;
38     src0_sel = 0;
39     src1_sel = 0;
40     imm_     = 0;
41 end
42
```

```
43 assign alu_op       = op;
44 assign imm          = imm_;
45 assign rf_ra0       = ra0;
46 assign rf_ra1       = ra1;
47 assign rf_wa        = wa;
48 assign rf_we        = we;
49 assign alu_src0_sel = src0_sel;
50 assign alu_src1_sel = src1_sel;
51
52 always @(*) begin
53     //R-type相关指令
54     if (inst[6:0] == 7'b0110011) begin
55         we       = 1;
56         ra0      = inst[19:15];
57         ra1      = inst[24:20];
58         wa       = inst[11:7];
59         src0_sel = 1'b0;
60         src1_sel = 1'b0;
61         imm_     = 32'h00000000;
62         case ({inst[31:25], inst[14:12]})
63             {7'b0000000, 3'b000}: op = `ADD;
64             {7'b0100000, 3'b000}: op = `SUB;
65             {7'b0000000, 3'b001}: op = `SLL;
66             {7'b0000000, 3'b010}: op = `SLT;
67             {7'b0000000, 3'b011}: op = `SLTU;
68             {7'b0000000, 3'b100}: op = `XOR;
69             {7'b0000000, 3'b101}: op = `SRL;
70             {7'b0100000, 3'b101}: op = `SRA;
71             {7'b0000000, 3'b110}: op = `OR;
72             {7'b0000000, 3'b111}: op = `AND;
73             default:              op = 5'b11111;
74         endcase
75     end
```

```
76      //I-type相关指令
77      else if (inst[6:0] == 7'b0010011) begin
78          we       = 1;
79          ra0      = inst[19:15];
80          ra1      = 5'b00000;
81          wa       = inst[11:7];
82          src0_sel = 1'b0;
83          src1_sel = 1'b1;
84          imm_     = {{20{inst[31]}},inst[31:25]};
85          case (inst[14:12])
86              3'b000: op = `ADD;
87              3'b010: op = `SLT;
88              3'b011: op = `SLTU;
89              3'b100: op = `XOR;
90              3'b110: op = `OR;
91              3'b111: op = `AND;
92              3'b001: begin
93                  op   = `SLL;
94                  imm_ = {{27{inst[24]}},inst[24:20]};
95              end
96              3'b101:
97              case (inst[31:25])
98                  7'b0000000:begin
99                      op   = `SRL;
100                     imm_ = {{27{inst[24]}},inst[24:20]};
101                 end
102                 7'b0100000:begin
103                     op   = `SRA;
104                     imm_ = {{27{inst[24]}},inst[24:20]};
105                 end
106                 default: op = 5'b11111;
107             endcase
108             default:op = 5'b11111;
```

```
109         endcase
110     end
111     //U-type相关指令(lui)
112     else if (inst[6:0] == 7'b0110111) begin
113         we        = 1;
114         ra0       = 5'b00000;
115         ra1       = 5'b00000;
116         wa        = inst[11:7];
117         src0_sel  = 1'b0;
118         src1_sel  = 1'b1;
119         imm_      = {inst[31:12],12'b0};
120         op        = `SRC1;
121     end
122     //U-type相关指令(auipc)
123     else if (inst[6:0] == 7'b0010111) begin
124             we        = 1;
125             ra0       = 5'b00000;
126             ra1       = 5'b00000;
127             wa        = inst[11:7];
128             src0_sel  = 1'b1;
129             src1_sel  = 1'b1;
130             imm_      = {inst[31:12],12'b0};
131             op        = `ADD;
132     end else begin
133             we        = 0;
134             ra0       = 5'b00000;
135             ra1       = 5'b00000;
136             wa        = 5'b00000;
137             src0_sel  = 1'b0;
138             src1_sel  = 1'b0;
139             imm_      = 32'b0;
140             op        = 5'b11111;
141     end
```

```
142 end
143 endmodule
```

根据指令 (inst),分别生成对应的 alu_op,imm,rf_ra0, rf_ra1,rf_wa, rf_we, alu_src0_sel, alu_src1_sel。

2.2 搭建 CPU

```
1  `include "./include/config.v"
2
3  module CPU (
4      input                   [ 0 : 0]            clk,
5      input                   [ 0 : 0]            rst,
6
7      input                   [ 0 : 0]            global_en,
8
9  /* ------------------------------ Memory (inst)
       ------------------------------ */
10     output                  [31 : 0]            imem_raddr,
11     input                   [31 : 0]            imem_rdata,
12
13 /* ------------------------------ Memory (data)
       ------------------------------ */
14     input                   [31 : 0]            dmem_rdata,
           // Unused
15     output                  [ 0 : 0]            dmem_we,
             // Unused
16     output                  [31 : 0]            dmem_addr,
           // Unused
17     output                  [31 : 0]            dmem_wdata,
           // Unused
18
19 /* ---------------------------------- Debug
       ---------------------------------- */
```

```
20      output                  [ 0 : 0]            commit,
21      output                  [31 : 0]            commit_pc,
22      output                  [31 : 0]            commit_inst
            ,
23      output                  [ 0 : 0]            commit_halt
            ,
24      output                  [ 0 : 0]
            commit_reg_we,
25      output                  [ 4 : 0]
            commit_reg_wa,
26      output                  [31 : 0]
            commit_reg_wd,
27      output                  [ 0 : 0]
            commit_dmem_we,
28      output                  [31 : 0]
            commit_dmem_wa,
29      output                  [31 : 0]
            commit_dmem_wd,
30
31      input                   [ 4 : 0]
            debug_reg_ra,
32      output                  [31 : 0]
            debug_reg_rd
33  );
34
35
36  // TODO
37  wire [31:0] cur_npc;
38  wire [31:0] cur_pc;
39  wire [31:0] cur_inst;
40
41  //仿真时用于控制global_en信号，在上板时删除
42  assign global_en=!(cur_inst==32'H00100073);
```

```
43
44 //例化PC_PLUS4模块
45 PC_PLUS4 pc_plus(
46 .pc(cur_pc),
47 .pc_plus4(cur_npc)
48 );
49
50 //例化PC模块
51 PC pc(
52 .clk    (clk),
53 .rst    (rst),
54 .en     (global_en),
55 .npc    (cur_npc),
56 .pc     (cur_pc)
57 );
58
59 //例化指令寄存器，从中读取指令
60 //可以不在cpu里例化，但实际没有影响
61 INST_MEM inst_mem(
62 //计算指令的位置
63 .a({{cur_pc-32'h00400000}/'d4}),    // input wire [8 : 0] a
64 .d(32'b0),      // input wire [31 : 0] d
65 .clk(clk),  // input wire clk
66 .we(1'b0),    // input wire we
67 .spo(cur_inst)  // output wire [31 : 0] spo
68 );
69
70 //定义decoder需要的变量
71 wire [4:0] rf_ra0;
72 wire [4:0] rf_ra1;
73 wire [4:0] rf_wa;
74 wire [31:0] rf_wd;
75 wire [4:0] alu_op;
```

```
76  wire alu_src0_sel;
77  wire alu_src1_sel;
78  wire [31:0] imm;
79  wire rf_we;
80  wire [31:0] rf_rd0;
81  wire [31:0] rf_rd1;
82
83  //例化decoder
84  DECODE decoder(
85  .inst(cur_inst),
86  .alu_op(alu_op),
87  .imm(imm),
88  .rf_ra0(rf_ra0),
89  .rf_ra1(rf_ra1),
90  .rf_wa(rf_wa),
91  .rf_we(rf_we),
92  .alu_src0_sel(alu_src0_sel),
93  .alu_src1_sel(alu_src1_sel)
94  );
95
96  //例化寄存器文件
97  REG_FILE reg_file(
98  .clk(clk),
99  .rf_ra0(rf_ra0),
100 .rf_ra1(rf_ra1),
101 .rf_wa(rf_wa),
102 .rf_we(rf_we),
103 .rf_wd(rf_wd),
104 .rf_rd0(rf_rd0),
105 .rf_rd1(rf_rd1),
106 .debug_reg_rd(debug_reg_rd),
107 .debug_reg_ra(debug_reg_ra)
108 );
```

```
109
110 //定义alu相关变量
111 wire [31:0] alu_src0;
112 wire [31:0] alu_src1;
113
114 //例化二选一数据选择器
115 MUX1 mux1(
116     .src0(rf_rd0),
117     .src1(cur_pc),
118     .sel(alu_src0_sel),
119     .res(alu_src0)
120 );
121 MUX1 mux2(
122     .src0(rf_rd1),
123     .src1(imm),
124     .sel(alu_src1_sel),
125     .res(alu_src1)
126 );
127
128 //例化alu
129 ALU alu(
130     .alu_src0(alu_src0),
131     .alu_src1(alu_src1),
132     .alu_op(alu_op),
133     .alu_res(rf_wd)
134 );
135
136 endmodule
```

在 cpu 中将所需的模块 (算数逻辑单元（ALU)、寄存器堆（RF)、程序计数器（PC)、译码器（DECODER)、多路选择器（MUX）等。) 例化并连接到对应线路上。

3 仿真结果与分析

仿真时，向指令寄存器内导入制定的 coe 文件，仿真所用的代码如下。

```
`timescale 1ns / 1ps
module tb_();
    reg clk;
    reg rst;
    wire [31:0] imem_raddr;
    wire dmem_we;
    wire [31 : 0]            dmem_addr;
    wire [31 : 0]            dmem_wdata;
    wire [ 0 : 0]            commit;
    wire [31 : 0]            commit_pc;
    wire [31 : 0]            commit_inst;
    wire [ 0 : 0]            commit_halt;
    wire [ 0 : 0]            commit_reg_we;
    wire [ 4 : 0]            commit_reg_wa;
    wire [31 : 0]            commit_reg_wd;
    wire [ 0 : 0]            commit_dmem_we;
    wire [31 : 0]            commit_dmem_wa;
    wire [31 : 0]            commit_dmem_wd;
    wire [31 : 0]            debug_reg_rd;
    //例化cpu模块并连接
    CPU cpu(
        .rst (rst),
        .clk (clk),
        .global_en(1'b1),
        .imem_rdata(0),
        .dmem_rdata(0),
        .debug_reg_ra(0),
        .imem_raddr(imem_raddr),
        .dmem_we(dmem_we),
        .dmem_addr(dmem_addr),
```

```
        .dmem_wdata(dmem_wdata),
        .commit(commit),
        .commit_pc(commit_pc),
        .commit_inst(commit_inst),
        .commit_halt(commit_halt),
        .commit_reg_we(commit_reg_we),
        .commit_reg_wa(commit_reg_wa),
        .commit_reg_wd(commit_reg_wd),
        .commit_dmem_we(commit_dmem_we),
        .commit_dmem_wa(commit_dmem_wa),
        .commit_dmem_wd(commit_dmem_wd),
        .debug_reg_rd(debug_reg_rd)
    );

    localparam CLK_PERIOD = 10;
    always #(CLK_PERIOD/2) clk=~clk;

    initial begin
        clk=0;
        rst=0;
        #10000
        $finish;
    end

endmodule
```

仿真结果如下:

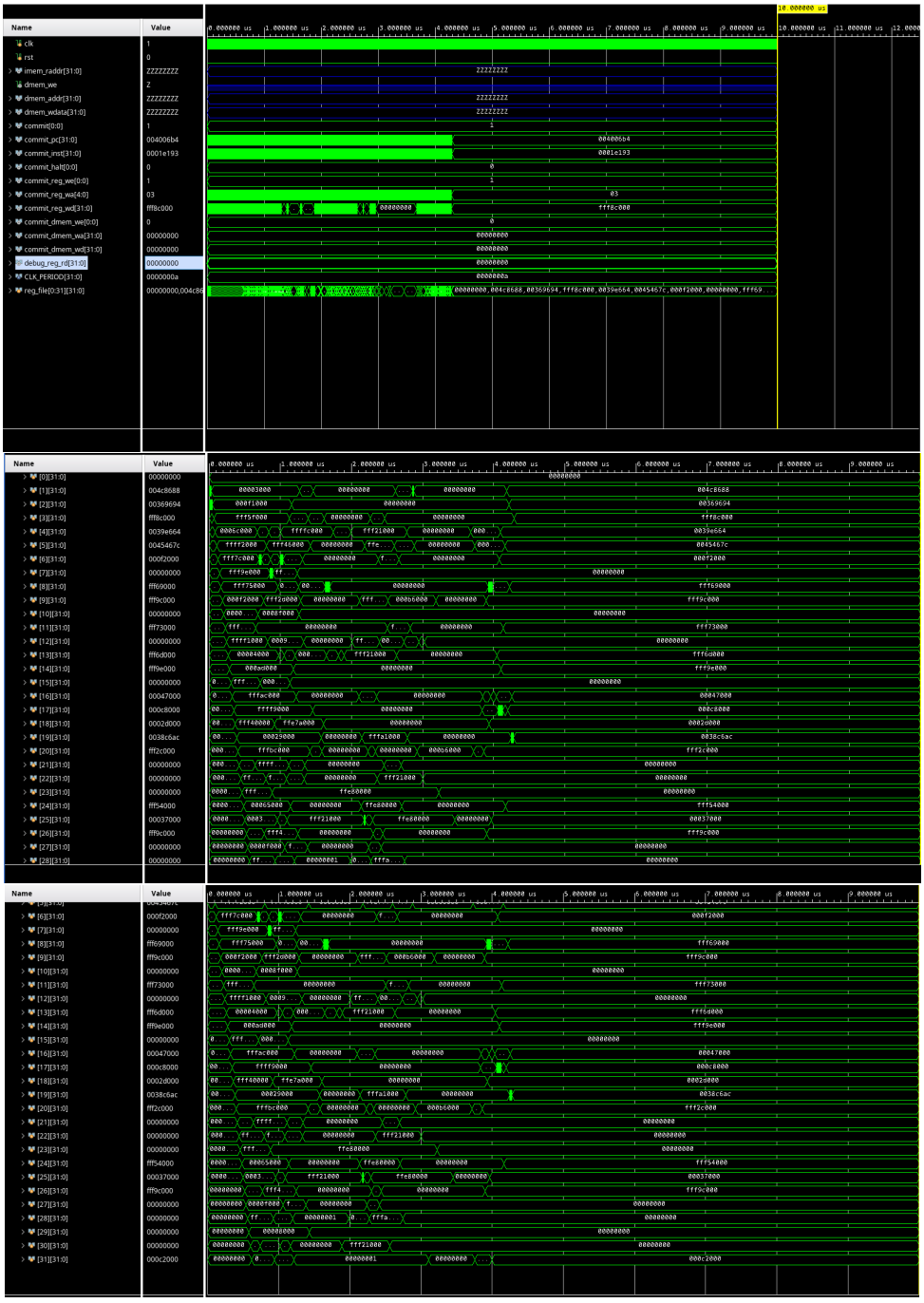

图 1: 仿真结果

汇编程序运行结果如下：

Name	Number	Value
zero	0	0x00000000
ra	1	0x004c8688
sp	2	0x00369694
gp	3	0xfff8c000
tp	4	0x0039e664
t0	5	0x0045467c
t1	6	0x000f2000
t2	7	0x00000000
s0	8	0xfff69000
s1	9	0xfff9c000
a0	10	0x00000000
a1	11	0xfff73000
a2	12	0x00000000
a3	13	0xfff6d000
a4	14	0xfff9e000
a5	15	0x00000000
a6	16	0x00047000
a7	17	0x000c8000
s2	18	0x0002d000
s3	19	0x0038c6ac
s4	20	0xfff2c000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0xfff54000
s9	25	0x00037000
s10	26	0xfff9c000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x000c2000
pc		0x004006bc

图 2: 运行结果

证明了仿真结果的正确性。

4 电路设计与分析

cpu 模块如图所示。

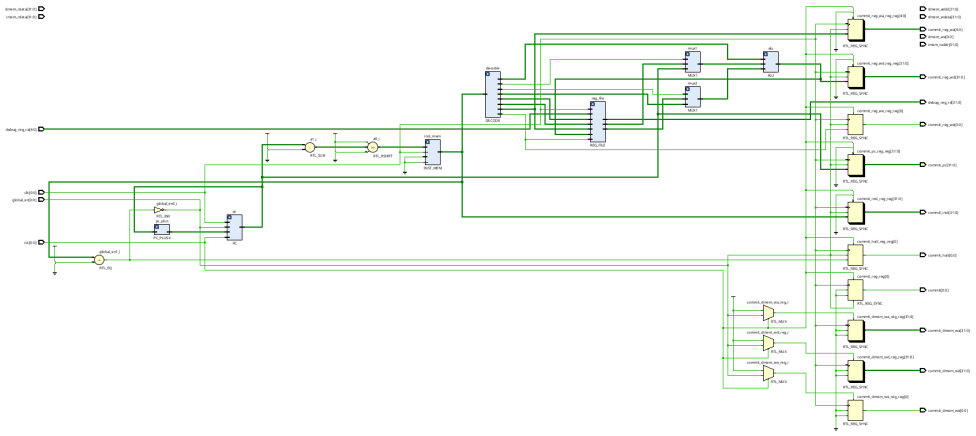

图 3: cpu 模块

5 测试结果与分析

上板结果如下：

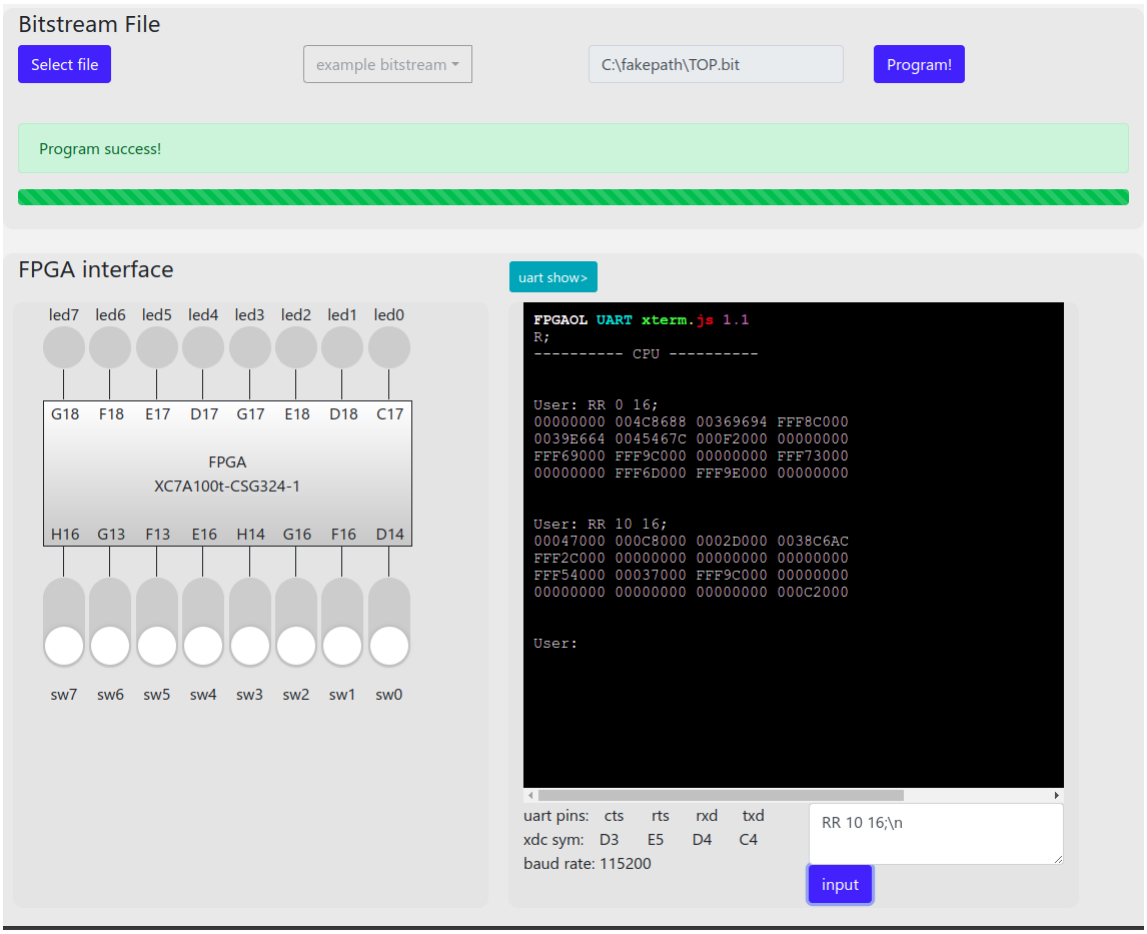

图 4: 测试结果

结果与实际运行结果相吻合。

6 思考与总结

6.1 本次实验的 CPU 中，哪些模块用到了时钟信号？

PC 模块，寄存器模块，指令存储器模块。

6.2 请分别给出一条指令，以符合下面的描述：

6.2.1 alu_src0 选择 pc

auipc x17, 0xf2

6.2.2 alu_src0 选择 rf_rd0

addi x1,x1,1

6.2.3 alu_src1 选择 rf_rd1

add x1,x1,x2

6.2.4 alu_src1 选择 imm

addi x1,x1,1

6.3 请指出本次实验的 CPU 中可能的关键路径；如果这条路径的延迟大于一个时钟周期，可能会带来什么影响？

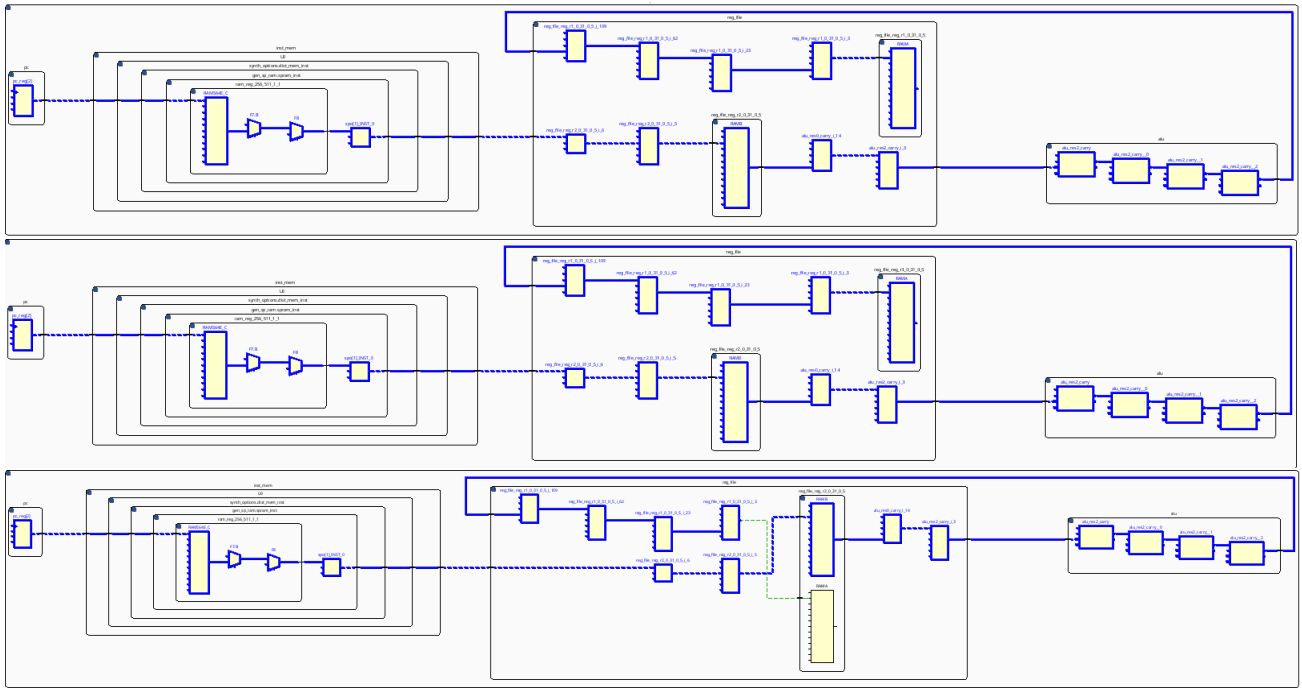

图 5: 可能的关键路径

若这条路径的延迟大于一个时钟周期，可能导致某些中间结果未能及时更新，进而使得后续操作基于错误或过时的数据进行，最终产生计算错误或指令执行失败。