Lab6 综合设计

李远航

PB20000137

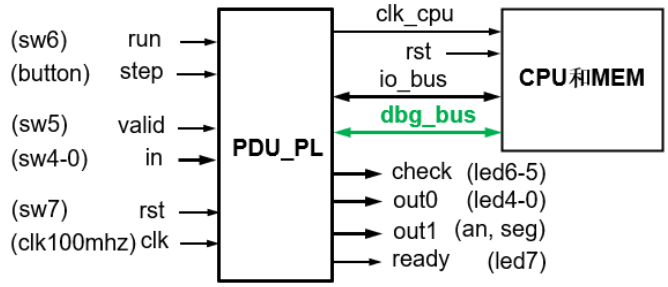
1. 实验目标

- 理解计算机硬件系统的组成结构和工作原理
- 掌握软硬件综合系统的设计和调试方法

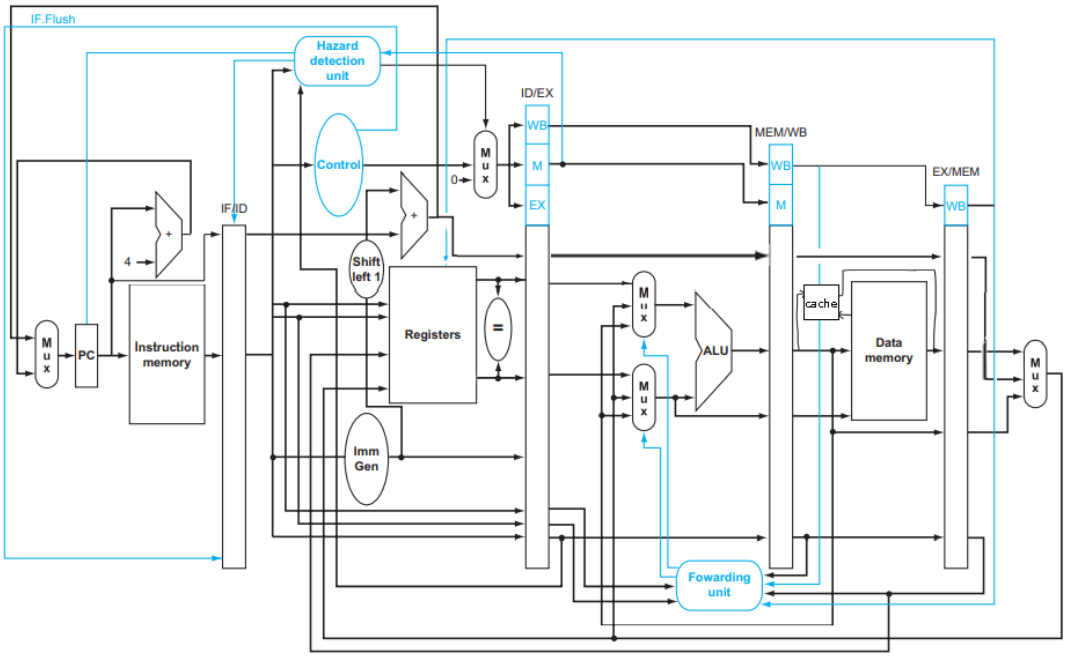
2. 实验环境

- PC 一台
- Vivado 2020.2
- fpgaol.ustc.edu.cn

3. 逻辑结构



4. 数据通路



在 CPU 流水线上增加 cache，简单修改 ALU 内部代码，使得其可以处理左移右移等运算

- 设计的 cpu 功能:
    - 实现 add, sub, and, or ,xor, sll, srl, slt, auipc, lw, sw
    - 实现 addi(nop), andi, ori, xori, slli, srli, slti
    - 实现 beq, blt, bne, bge
    - 实现 jal, jalr
- 实现全相联 cache

5. 具体实现

- alu 内部修改:

```
module alu(
    input  [31:0] a, b,
    input  [2:0] f,
    output reg z_r,
    output reg [31:0] y_r
);
    always@(*)begin
        case(f)
            3'b000: y_r = a + b;
            3'b001: y_r = a - b;
            3'b010: y_r = a & b;
            3'b011: y_r = a | b;
            3'b100: y_r = a ^ b;
            3'b101: y_r = a << b;
            3'b110: y_r = a >> b;
            3'b111: begin
                if (a<b)
                    y_r = 1;
                else
                    y_r = 0;
            end
            default: y_r = 0;
        endcase
        if (y_r == 0)
            z_r = 1;
        else
            z_r = 0;
    end
endmodule
```

- cache 的实现

使用全相联 cache(2^3*2^40)，利用地址的 9~2 位来对应 cache

```
wire [31:0] memory_read_infact;
reg [40:0] cache[0:7];
integer i;
wire Hit;
```

```
assign Hit = (MemRead_MEM)&&cache[alu_result_MEM[11:8]][40]&&
(alu_result_MEM[9:2]==cache[alu_result_MEM[11:8]][39:32]);
always@(negedge clk or posedge rst) begin
    if (rst) begin
        for (i = 0 ; i < 8 ; i = i + 1)
            cache[i][40:0] <= 41'b0;
    end
    else begin
        if(Hit)
            memory_read_data <= cache[alu_result_MEM[11:8]][31:0];
        else if(MemRead_MEM) begin
            memory_read_data <= memory_read_infact;
            cache[alu_result_MEM[11:8]][31:0] <= memory_read_infact;
            cache[alu_result_MEM[11:8]][39:32] <= alu_result_MEM[9:2];
            cache[alu_result_MEM[11:8]][40] <= 1;
        end
        else if(MemWrite) begin
            cache[alu_result_MEM[11:8]][31:0] <= Read_data_2_MEM;
            cache[alu_result_MEM[11:8]][39:32] <= alu_result_MEM[9:2];
            cache[alu_result_MEM[11:8]][40] <= 1;
        end
    end
end
```

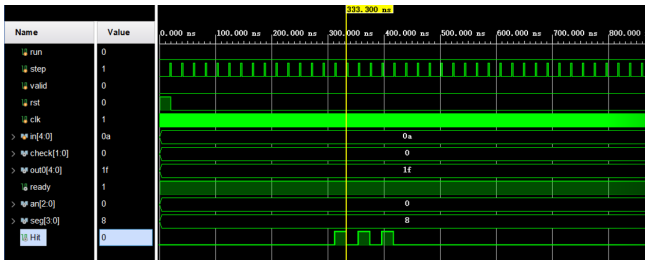
6. 运行截图

- cache 的访问

  使用如下测试程序:

  ```
  .text
      addi x1, x0, 10
      sw x1, 0x300(x0)
      andi x2, x0, 15
      sw x2, 0x304(x0)
      ori x3, x0, 20
      sw x3, 0x308(x0)
      nop
      nop
      nop
      nop
      nop
      lw x1, 0x308(x0)
      nop
      lw x2, 0x304(x0)
      nop
      lw x3, 0x300(x0)
      nop
  ```

  观察波形，可以观测到，cache 访问命中

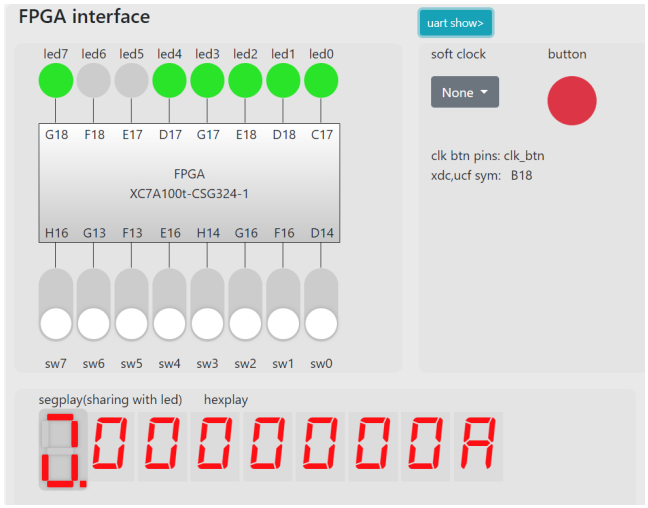


- 额外指令的测试:

  使用如下测试汇编

```
.text
    addi x2, x0, 0
pretest:
    addi x1, x0, 1 # out = 1
    addi x1, x1, 1 # out = 2
    slli x1, x1, 1 # out = 4
    srli x1, x1, 1 # out = 2
    ori x1, x1, 1 # out = 3
    andi x1, x1, 2 # out = 2
    xori x1, x1, 1 # out = 3
    sw x1, 0x408(x0) # out = 3
    slti x1, x1, 4 # out = 1
    add x1, x1, x1 # out = 2
    addi x2, x2, 1
    sw x2, 0x408(x0) # out = x2
    blt x2, x1, pretest
    bne x2, x1, pretest
    addi x2, x0, 10
    sw x2, 0x408(x0) # out = 10
    bge x1, x2, pretest
```

  上板烧写测试，符合设计



7. 实验收获

- 熟悉了 cache 的原理和设计
- 进一步理解了流水线 CPU 的结构和工作原理
- 对流水线的冒险处理有的更深的认识