中国科学技术大学计算机学院

《计算机组成原理实验》报告

实验题目：综合设计

学生姓名：李昱祁

学生学号：PB18071496

完成日期：2020/07/08

计算机实验教学中心制

2019 年 09 月

【实验目的】

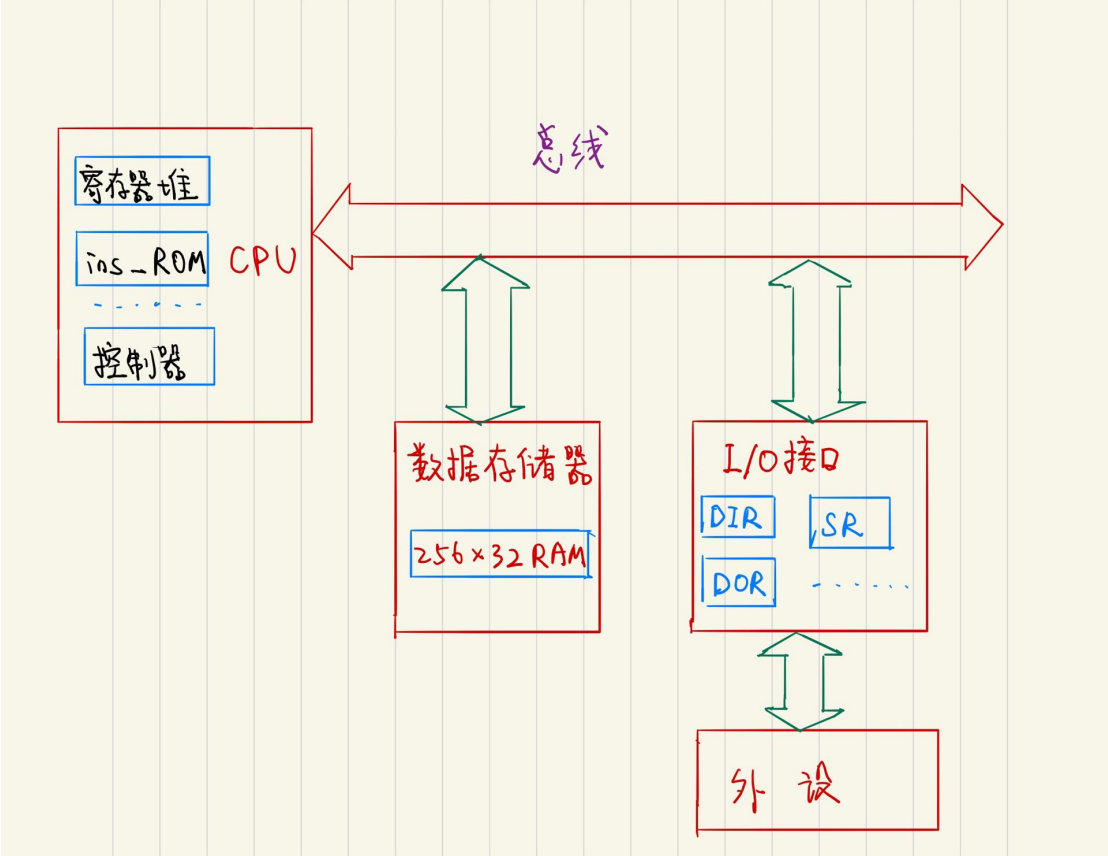
1. 理解计算机系统的组成结构和工作原理；
2. 理解计算机总线和接口的结构和功能；
3. 掌握软硬件综合系统的设计和调试方法。

【实验环境】

1. Ubuntu 18.04 操作系统
2. Vivado 2019.1

【实验过程】

计算机硬件组成设计如下：



主要讲述实验中设计的 5 个部分，分别是：

总线控制器、
I/O 接口、

CPU 的修改、
汇编代码、
仿真文件的设计

一.　总线控制器模块

1. 逻辑设计：

本次计算机设计中，所有硬件组成通过总线相连。个人设计的总线共 100 位，其中：

a. 地址线 32 位，用于控制读、写存储器以及进行 I/O 操作；
b. 数据线 66 位，其中 CPU 输出至存储器、I/O 接口 32 位，用于写存储器、I/O 输出；输入至 CPU 34 位，包括 32 位数据（存储器、I/O 接口都可以占用），以及 2 位来自 I/O 接口的外设状态信息
c. 控制线 2 位，用于控制存储器写使能、I/O 接口的 DOR 是否更新

2. 核心代码：

（因为总线 bus 是顶层模块，所以外设的输入信息、输出至外设的信息、外设的状态信息等，也接入至该模块完成）

```
module bus
    (
        input clk,rst,
        input [1:0] is_ready,
        input [31:0] data_input,
        output data_output
    );

    /*
        总线控制器模块

        100 位 总线

        d_bus_1  表示输入至 cpu 的数据总线分支
        d_bus_2     cpu 输出 的数据线
        c_bus    控制总线
        a_bus    地址总线
```

```
*/

wire [33 : 0] d_bus_1,
wire [31 : 0] d_bus_2;
wire [31 : 0] a_bus;
wire [ 1 : 0] c_bus;

wire [31:0] data_IO, data_MEM;
wire we_1,we_2;

cpu CPU
(
    .d_bus_1(d_bus_1),
    .d_bus_2(d_bus_2),
    .c_bus(c_bus),
    .a_bus(a_bus),
    .clk(clk),
    .rst(rst)
);

IO_interface IO
(
    .clk(clk),
    .is_ready_1(is_ready[0]),
    .is_ready_2(is_ready[1]),
    .data_input(data_input),    //来自于输入外设
    .data_cpu(d_bus_2),
    .we(we_2),
    .CR(c_bus),

    .DIR(data_IO),            //可输出至总线，供 CPU 使用
    .SR(d_bus_1[33:32]),
    .data_output(data_output)
);

mem MEM(
    .addr(a_bus),
    .clk(clk),
    .we(we_1),
    .r_data(data_MEM),
    .w_data(d_bus_2)
);

// 确定 I/O 输入与 存储器读 谁占据数据线
```

```
    assign d_bus_1[31:0] = a_bus > 32'h0000_0400 ? data_IO : data_MEM;

    // 此 we 控制数据存储器是否可写
    assign we_1 = (c_bus == 2'b10) & (a_bus <= 32'h0000_0400);

    // 此 we 控制是否要把数据写入 io 接口的 DOR
    assign we_2 = (c_bus == 2'b10) & (a_bus > 32'h0000_0400);

endmodule
```

二.　I/O 接口模块

1. 逻辑设计：

以下各项功能均在时钟边沿完成：

a. 总线上“输出”控制信号有效时，将总线上的数据存入数据输出寄存器 DOR 中
b. 在输出外设准备好后，将 DOR 中的数据输出至外设
c. 在输入外设准备好后，将外设中的数据存放至数据输入寄存器 DIR
d. 将 DIR 中的数据返回至总线控制器
e. 将输入、输出外设的状态信息返回至总线

2. 核心代码：

```
module IO_interface
    (
        input clk,
        input is_ready_1,
        input is_ready_2,
        input [31:0] data_input,
        input [31:0] data_cpu,
        input we,
```

```
    input [1:0] CR,

    output reg [31:0] data_output,
    output reg [31:0] DIR,
    output reg [1:0] SR

);

/*
*  is_ready_1 表示 I/O 输入是否已经准备就绪
*  is_ready_2 表示 I/O 输出是否已经准备就绪
*  data 表示从外设接收到的数据
*/

reg [31:0] DOR = 0;

always@(posedge clk)
begin
    SR[0] = is_ready_1;
    SR[1] = is_ready_2;

    if(we)
    begin
        DOR = data_cpu;
    end

    if(is_ready_1)
    begin
        DIR = data_input;
    end

    if(is_ready_2)    //若输出外设已准备好，那么将 DOR 中的数据输出
    begin
        data_output = DOR;
    end

end


endmodule
```

三. CPU 中的 I/O 等待

1. 设计原因：

在 I/O 设备进行工作时，不一定会立即完成；并且在设计本次实验的汇编代码时，并没有将“轮询”使用汇编代码完成。因此，CPU 可能需要暂停工作（不更新 PC），以等待 I/O 任务完成

2. 修改的代码：

在进行 I/O 时，若需要输入但输入设备未准备完成，或者需要输出但输出设备未准备好，这两种情况下 CPU 均会暂停，直至外设准备就绪：

```
    wire nop;
// 判断是否要等待 I/O
    assign nop = ( (alu_result > 32'h0000_0400) &
                ( (signal[3] & ~d_bus_1[32]) | (signal[7] & ~d_bus_1[33]) ) );
```

之后，需要在时钟边沿更新 pc 时对 nop 进行判断：

```
/* 异步复位以及更新 pc */
    always @(posedge clk or posedge rst)
    begin
        if(rst)
        begin
            pc = 32'h00000000;
        end
        else if(!nop)
        begin
            pc = pc_new;
        end
    end
```

四. 斐波那契序列计算

1. 逻辑设计：

完成数据输入（F0、F1）后，设计一循环，输出该斐波那契数列的前 20 项

2. 汇编代码：

首先在 t4、t5 中装入指向存储器映像地址的指针，其中 1025 = 0x401 表示数据输入寄存器 DIR 的地址，1026 = 0x402 表示数据输出寄存器 DOR 的地址

之后根据 t4、t5 中的地址，即可通过 lw、sw 指令完成 I/O 操作

其余的，如循环等的实现等，不在此处赘述。

```
main:

    # t4 指向 IO 输入
    # t5 指向 IO 输出
    addi $t4,$0,1025
    addi $t5,$0,1026

    # 读取 f0 、f1，放入 t0、t1
    lw $t0,0($t4)
    lw $t1,0($t4)

    addi $t3,$0,10

loop:
    add $t0,$t0,$t1
    sw $t0,0($t5)

    add $t1,$t0,$t1
    sw $t1,0($t5)

    addi $t3,$t3, -1
    beq  $t3, $0, finish

    j loop

finish:
    j finish
```

此次使用 MARS 将汇编代码转换为 16 进制机器码。需要注意的是，MARS 中代码段（.text）从地址 0x0040_0000 开始，因此跳转指令 j 的地址，在代码移植到本实验的指令 ROM 中时，需要做相应的修改。

五. 仿真文件

1. 设计思路：

a. 产生时钟、复位信号（仅在开始时复位一次）
b. 输入外设的 ready 位在一段时间内不有效，用于检查 CPU 是否能正确等待 I/O；输出外设 ready 位一直有效
c. 用仿真信号，模拟外设的输入数据（data_input）

2. 仿真文件：

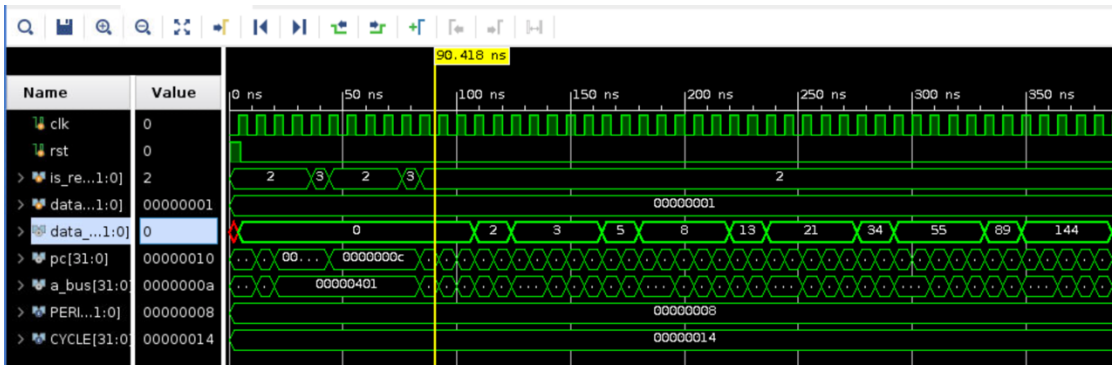
内容如下：

```
module fibo_tb;

    reg clk;
    reg rst;
    reg [1:0] is_ready;
    reg [31:0] data_input;

    wire [31:0] data_output;

    parameter PERIOD = 8;
    parameter CYCLE = 20;

    initial
    begin
        clk = 0;
        repeat (5 * CYCLE)
            #(PERIOD/2) clk = ~clk;
        $finish;
    end

    initial
    begin
        rst = 1;
        #(8)   rst = 0;

    end

    initial
    begin
        is_ready[0] = 0;
        #(16) is_ready[0] = 1;
        #(8)  is_ready[0] = 0;

        #(32) is_ready[0] = 1;
```

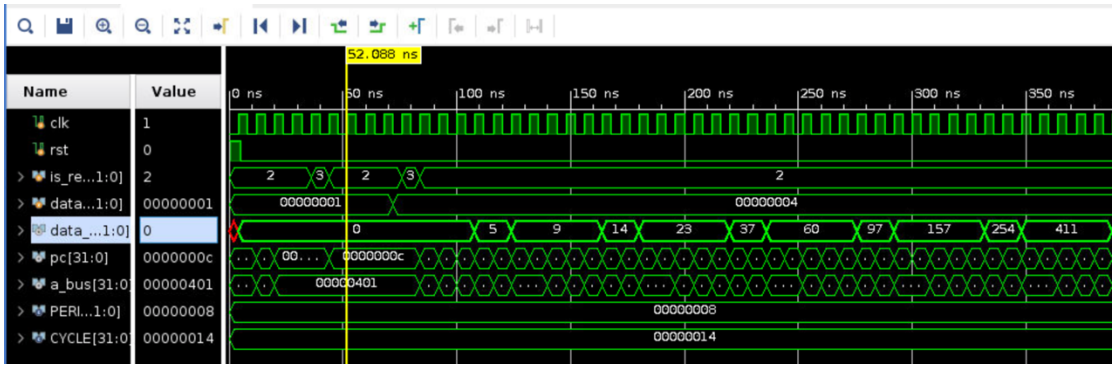
```
        #(8)  is_ready[0] = 0;
    end

    initial
    begin
        is_ready[1] = 1;
    end

    initial
    begin
        data_input = 32'h0000_0001;
    end



    bus BUS
    (
        .clk(clk),
        .rst(rst),


        .is_ready(is_ready),
        .data_input(data_input),
        .data_output(data_output)
    );

endmodule
```

【仿真结果】

在 vivado 中进行仿真，结果如下所示：
首先令 F0 = 1, F1 = 1



再令 F0 = 1, F1 = 4



通过上述两个仿真结果可以发现，系统正确的按顺序输出了 F2、F3、F4……；在进行输入时，若外设的 ready 位无效，则会产生中断，pc 值不会更改，CPU 等待 I/O 完成

【总结与思考】

1. 回顾了***组成原理*** 课程后半部分所讲的总线、I/O 接口的相关知识，亲自设计并仿真了二者，熟悉了其工作原理
2. 利用中断实现了 I/O 操作，使本课程实验中设计的计算机系统功能更完善
3. 熟悉了汇编语言的使用，以及 MIPS 体系结构的相关知识

【改进建议】

本学期情况特殊，综合实验仅通过仿真完成，故无意见。

【附录】

“综合设计实验视频.mov”