计算机系统实验课程报告

——CPU 改造、下板实验

院　　系　计算机科学与技术学院

专　　业　计算机科学与技术

姓　　名　张宇

学　　号　2251745

指导老师　郭玉臣

完成日期　2025 年 4 月 10 日

目录

一、 实验环境与实验内容

1.1 实验环境

操作系统	Windows 11
开发环境	Vivado 2016.2
硬件配置	XILINX NEXYS 4 DDR

1.2 实验内容

本次实验通过数码管实验、Flash 读取实验以及 AXI 通信实验入手，在龙芯 CPU LS132R 上运行自己编写的测试程序。我使用 C 语言和 MIPS 指令分别编写了性能验证程序。C 语言程序利用 gcc 编译器编译生成目标程序，而 MIPS 指令汇编程序则通过 Mars 编译生成目标程序。此外，本次实验中对两种方式下 CPU 的定点运算性能进行了测试、比较和分析。

在将龙芯 CPU LS132R 移植到 Nexys 4 开发板上的过程中，首先需要将 C 语言代码程序烧录进 flash 中，然后烧录 FPGA 程序。当按下开发板上的复位键时，LS132R 核会首先进入固定地址取指令，通常是从 flash 上取指令。启动过程中，flash 程序会将数据和其他必要内容搬运到适当的 RAM 中，完成整个初始化过程，然后进入 main 函数运行 C 语言程序。

二、实验过程与方法

1. 建立新的 vivado 工程文件，将《自己动手写 CPU》第 11 章的所有文件加载入。该 CPU 工程文件主要分成三个部分：第一，头文件部分，第二，CPU 主体部分，第三，仿真部分。在载入到本地 vivado 工程中时，需要分清以上三个模块，分别载入。

2. 将测试指令用 Mars 软件转化为 16 进制 coe 文件，并配置 ip 核以便后续进行下板测试。

3. 修改底层模块，比如分支预测部分，在《自己动手写 CPU》的代码当中，连续两个分支指令，尤其是一条条件跳转指令，紧接着一条无条件跳转指令会出现一定的问题。如下图所示，bgtz 在计算出是否应当分支的时候，其目标地址已经被 j 指令的目标地址给覆盖掉，表现为 bgtz 指令无效。因此我们可以修改分支预测部分或者更新测试文件。

```
bgtz $10,TAG5
j test7_error
```

比如加上 break 指令，对于《自己动手写 CPU》的代码而言，是不存在 break 指令的（查阅了纸质书），代码中执行 break 的逻辑是走 inst_invalid 的路线。而这个操作会将 cause 寄存器赋值为 0x28，而非我们检验所需的 0x24。因此我们可以自行加上 break 指令，或者更新测试文件。

4. 修改顶层模块，配置并增添分频器和数码管显示模块，使得下板测试的时候可以实时显示，方便评测。

5. 修改约束文件，使之符合我们的 Nexys 4 开发板的引脚，使得下板测试的时候可以实时显示，方便评测。

6. 增添仿真文件，连接 MIPS 流水线 CPU 模块和数码管模块，对测试指令进行一条条连续的测试。

7. 完成以上部署后，进行仿真模拟，该 CPU 采用了读文件的方式读取指令，对应代码为 inst_ram.v 文件，将其读写的文件改为自己的指令即可。

8. 然后进行综合，生成二进制 bit 流，下板调试，根据测试文件的对应结果检验 MIPS 流水线 CPU 是否已经修改成功。

三、程序修改说明

3.1 顶层模块

仿照我们上学期在《计算机系统结构》实验中的顶层模块设计，在顶层模块中将 CPU、指令存储器、数据存储器、七段数码显示管模块相连接。注意，顶层模块同时也包含了分频器模块的内容。

代码如下：

```
`include "defines.v"

module openmips_min_sopc(

    input wire clk,
    input wire  rst,
    output [7:0] O_Seg,
    output [7:0] O_Sel

);

  reg clk_s;
  initial clk_s=0;
  integer cnt=0;
  always @(posedge clk)
  begin
  if(cnt < 10000)
    cnt = cnt+1;
  else
     begin
```

```
        cnt = 0;
        clk_s = ~clk_s;
    end
end


//连接指令存储器
wire[`InstAddrBus] inst_addr;
wire[`InstBus] inst;

wire rom_ce;
wire mem_we_i;
wire[`RegBus] mem_addr_i;
wire[`RegBus] mem_data_i;
wire[`RegBus] mem_data_o;
wire[3:0] mem_sel_i;
wire mem_ce_i;
wire[5:0] int;
wire timer_int;
wire[`DataBus] Seg7_In;

//assign int = {5'b00000, timer_int, gpio_int, uart_int};
assign int = {5'b00000, timer_int};

  openmips openmips0(
        .clk(clk_s),
        .rst(rst),

        .rom_addr_o(inst_addr),
        .rom_data_i(inst),
        .rom_ce_o(rom_ce),

        .int_i(int),

        .ram_we_o(mem_we_i),
        .ram_addr_o(mem_addr_i),
        .ram_sel_o(mem_sel_i),
        .ram_data_o(mem_data_i),
        .ram_data_i(mem_data_o),
        .ram_ce_o(mem_ce_i),

        .timer_int_o(timer_int)

  );

  inst_rom inst_rom0(
        .ce(rom_ce),
        .addr(inst_addr),
        .inst(inst)
  );

  data_ram data_ram0(
        .clk(clk_s),
        .ce(mem_ce_i),
        .we(mem_we_i),
```

```
        .addr(mem_addr_i),
        .sel(mem_sel_i),
        .data_i(mem_data_i),
        .data_o(mem_data_o),
        .debugreg(Seg7_In)
    );

    Seg7x16 Seg7(clk, rst, 1, Seg7_In, O_Seg, O_Sel);

endmodule
```

3.2 控制器模块

为了配合测试文件的要求，即中断例程起始地址为 0x00400004，需要修改 ctrl.v 文件里的跳转地址， 将中断跳转地址更改为要求的 0x00400004。这样可以保证最后测试输出结果的正确性。

代码如下：

```
`include "defines.v"

module ctrl(

    input wire                  rst,
    input wire[31:0]            excepttype_i,
    input wire[`RegBus]         cp0_epc_i,
    input wire                  stallreq_from_id,

  //来自执行阶段的暂停请求
    input wire                  stallreq_from_ex,

    output reg[`RegBus]         new_pc,
    output reg                  flush,
    output reg[5:0]             stall

);


    always @ (*) begin
        if(rst == `RstEnable) begin
            stall <= 6'b000000;
            flush <= 1'b0;
            new_pc <= `ZeroWord;
        end else if(excepttype_i != `ZeroWord) begin
          flush <= 1'b1;
          stall <= 6'b000000;
            case (excepttype_i)
                32'h00000001:       begin   //interrupt
                    new_pc <= 32'h00000020;
                end
                32'h00000008:       begin   //syscall
                    new_pc <= 32'h00400004;
                end
                32'h0000000a:       begin   //inst_invalid
                    new_pc <= 32'h00400004;
```

```
                end
                32'h0000000d:       begin    //trap
                    new_pc <= 32'h00400004;
                end
                32'h0000000c:       begin    //ov
                    new_pc <= 32'h00400004;
                end
                32'h0000000e:       begin    //eret
                    new_pc <= cp0_epc_i;
                end
                default : begin
                end
            endcase
        end else if(stallreq_from_ex == `Stop) begin
            stall <= 6'b001111;
            flush <= 1'b0;
        end else if(stallreq_from_id == `Stop) begin
            stall <= 6'b000111;
            flush <= 1'b0;
        end else begin
            stall <= 6'b000000;
            flush <= 1'b0;
            new_pc <= `ZeroWord;
        end    //if
    end      //always


endmodule
```

3.3 七段数码管驱动模块

七段数码管是 Nexys4 开发板上用于显示数值的一个部件，我们需要将指定的 0x10010000 内存单元最终结果显示到七段数码管上，以验证 CPU 改造成功。因此需要添加七段数码管的驱动模块。

沿用在 MIPS54 条指令动态流水线项目中的七段数码管显示模块即可。

代码如下：

```
`timescale 1ns / 1ns

module Seg7x16(
    input Clk,
    input Reset,
    input Cs,
    input [31:0] I_Data,
    output [7:0] O_Seg,
    output [7:0] O_Sel
    );

    reg [14:0] Cnt;
    always @ (posedge Clk, posedge Reset)
      if (Reset)
        Cnt <= 0;
      else
```

```
        Cnt <= Cnt + 1'B1;

    wire Seg7_Clk = Cnt[14];

     reg [2:0] Seg7_Addr;

     always @ (posedge Seg7_Clk, posedge Reset)
       if(Reset)
          Seg7_Addr <= 0;
        else
          Seg7_Addr <= Seg7_Addr + 1'B1;

     reg [7:0] O_Sel_R;

     always @ (*)
       case(Seg7_Addr)
          7 : O_Sel_R = 8'B01111111;
          6 : O_Sel_R = 8'B10111111;
          5 : O_Sel_R = 8'B11011111;
          4 : O_Sel_R = 8'B11101111;
          3 : O_Sel_R = 8'B11110111;
          2 : O_Sel_R = 8'B11111011;
          1 : O_Sel_R = 8'B11111101;
          0 : O_Sel_R = 8'B11111110;
        endcase

     reg [31:0] I_Data_Store;
     always @ (posedge Clk, posedge Reset)
       if(Reset)
          I_Data_Store <= 0;
        else if(Cs)
          I_Data_Store <= I_Data;

     reg [7:0] Seg_Data_R;
     always @ (*)
       case(Seg7_Addr)
          0 : Seg_Data_R = I_Data_Store[3:0];
          1 : Seg_Data_R = I_Data_Store[7:4];
          2 : Seg_Data_R = I_Data_Store[11:8];
          3 : Seg_Data_R = I_Data_Store[15:12];
          4 : Seg_Data_R = I_Data_Store[19:16];
          5 : Seg_Data_R = I_Data_Store[23:20];
          6 : Seg_Data_R = I_Data_Store[27:24];
          7 : Seg_Data_R = I_Data_Store[31:28];
        endcase

     reg [7:0] O_Seg_R;
     always @ (posedge Clk, posedge Reset)
       if(Reset)
          O_Seg_R <= 8'Hff;
        else
          case(Seg_Data_R)
            4'H0 : O_Seg_R <= 8'HC0;
          4'H1 : O_Seg_R <= 8'HF9;
          4'H2 : O_Seg_R <= 8'HA4;
```

```
            4'H3 : O_Seg_R <= 8'HB0;
            4'H4 : O_Seg_R <= 8'H99;
            4'H5 : O_Seg_R <= 8'H92;
            4'H6 : O_Seg_R <= 8'H82;
            4'H7 : O_Seg_R <= 8'HF8;
            4'H8 : O_Seg_R <= 8'H80;
            4'H9 : O_Seg_R <= 8'H90;
            4'HA : O_Seg_R <= 8'H88;
            4'HB : O_Seg_R <= 8'H83;
            4'HC : O_Seg_R <= 8'HC6;
            4'HD : O_Seg_R <= 8'HA1;
            4'HE : O_Seg_R <= 8'H86;
            4'HF : O_Seg_R <= 8'H8E;
            endcase

      assign O_Sel = O_Sel_R;
      assign O_Seg = O_Seg_R;

endmodule
```

四、约束文件修改说明

对于约束文件的修改较为容易。根据测试文件的需求，我们需要从 Nexys 4 开发板上得到的选择信号，以及时钟和复位信号，并且将指定的 0x10010000 内存单元最终结果输出到七段数码管上，因此需要驱动数码管的引脚。

根据上述需求，编写约束文件如下：

```
set_property PACKAGE_PIN E3 [get_ports clk]
set_property PACKAGE_PIN N17 [get_ports rst]

set_property PACKAGE_PIN T10 [get_ports {O_Seg[0]}]
set_property PACKAGE_PIN R10 [get_ports {O_Seg[1]}]
set_property PACKAGE_PIN K16 [get_ports {O_Seg[2]}]
set_property PACKAGE_PIN K13 [get_ports {O_Seg[3]}]
set_property PACKAGE_PIN P15 [get_ports {O_Seg[4]}]
set_property PACKAGE_PIN T11 [get_ports {O_Seg[5]}]
set_property PACKAGE_PIN L18 [get_ports {O_Seg[6]}]
set_property PACKAGE_PIN H15 [get_ports {O_Seg[7]}]

set_property PACKAGE_PIN J17 [get_ports {O_Sel[0]}]
set_property PACKAGE_PIN J18 [get_ports {O_Sel[1]}]
set_property PACKAGE_PIN T9 [get_ports {O_Sel[2]}]
set_property PACKAGE_PIN J14 [get_ports {O_Sel[3]}]
set_property PACKAGE_PIN P14 [get_ports {O_Sel[4]}]
set_property PACKAGE_PIN T14 [get_ports {O_Sel[5]}]
set_property PACKAGE_PIN K2 [get_ports {O_Sel[6]}]
set_property PACKAGE_PIN U13 [get_ports {O_Sel[7]}]

set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[4]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

create_clock -period 200.000 -name clk_pin -waveform {0.000 100.000}
[get_ports clk]
set_input_delay -clock [get_clocks *] 1.000 [get_ports rst]
set_output_delay -clock [get_clocks *] 0.000 [get_ports -filter { NAME =~
"*" && DIRECTION == "OUT" }]
```

五、仿真分析

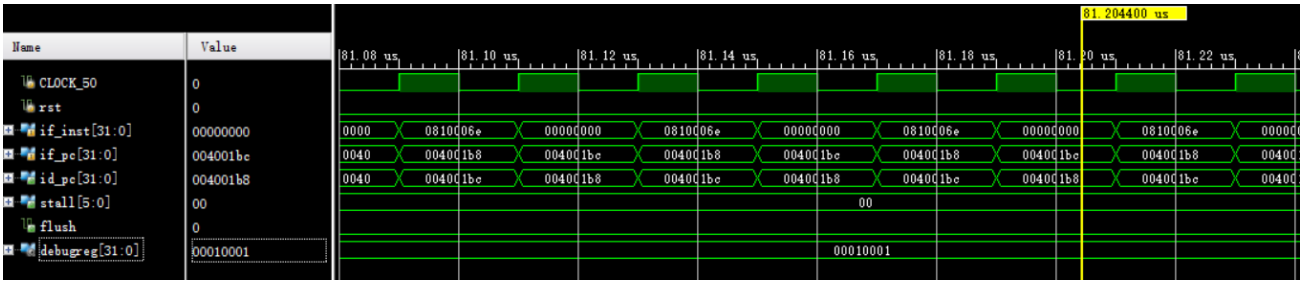
5.1 仿真模块

使用如下文件进行仿真：

```
`include "defines.v"
`timescale 1ns / 1ps

module openmips_min_sopc_tb();
  reg     CLOCK_50;
  reg     rst;

  initial begin
    CLOCK_50 = 1'b0;
    forever #10 CLOCK_50 = ~CLOCK_50;
  end

  initial begin
    rst = `RstEnable;
    #195 rst= `RstDisable;
  end

  openmips_min_sopc openmips_min_sopc0(
    .clk(CLOCK_50),
    .rst(rst)
  );

endmodule
```

5.2 仿真分析

仿真分析展现了流水线能够正常流动，我们可以看到如下的仿真波形图：



六、下板验证

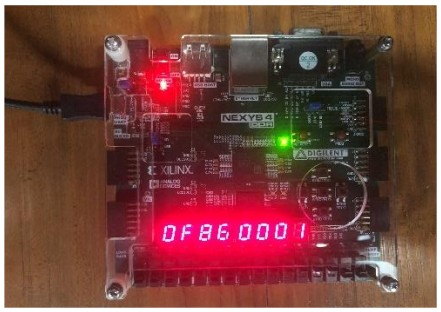
6.1下板测试过程

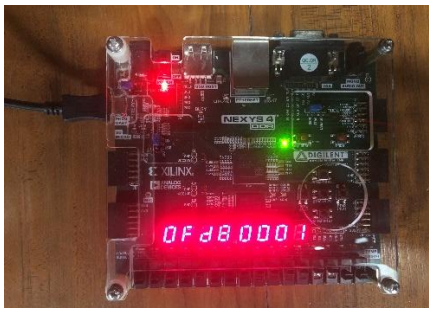
1. 编写 16 进制的七段数码管驱动程序；
2. 对时钟信号进行分频，保证能够在板子上看到清晰的结果；
3. 将七段数码管驱动程序与 89 条指令动态流水线 CPU 在顶层模块做连接；
4. 进行综合，编写管脚约束文件；
5. 生成二进制 bit 流，下板。

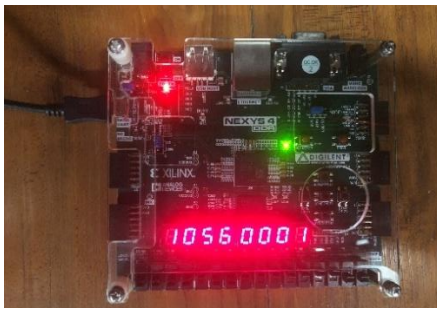
通过正确的文件约束，使板子上依次显示某一寄存器中的值，从而验证程序的正确性。下板进行测试，依次展示七段数码管选择的 debug 寄存器的数值。

6.2 结果检验展示

下面截取部分图片证明 89 条流水线的正确性。可以发现，数码管低半字显示 0x0001，高半字随着时钟中断而计数，符合测试正确结果预期。







七、实验体会

在本次的 MIPS 指令集流水线 CPU 实验中，我对 CPU 的运行有了更全面的认识。本次实验内容为更改现有 CPU 使之能够在我们的 Nexys 4 开发板上正常运行。实验本身需要的代码量并不是很大，但与自己动手写 CPU 不一样的是——改造现有 CPU 需要阅读别人的程序，完

全搞明白别人的代码功能与代码逻辑以及各模块之间的连接关系，某种意义上来说这是更加困难的一件事情。因为这需要对 MIPS 指令集流水线 CPU 的原理有一个更好的理解。

经过实践，我认为可以慢慢理清思路，针对测试数据对各个模块逐个击破，在修改单个模块的同时，也要考虑这一模块对全局的影响，比如控制信号等是否需要修改。比如修改分支预测部分，在《自己动手写 CPU》的代码当中，连续两个分支指令，尤其是一条条件跳转指令，紧接着一条无条件跳转指令会出现一定的问题。例如 bgtz 在计算出是否应当分支的时候，其目标地址已经被 j 指令的目标地址给覆盖掉，表现为 bgtz 指令无效。因此我们可以修改分支预测部分或者更新测试文件。

另外，我们还需要注意一些 debug 上的小技巧。对于硬件程序，波形图仿真验证、生成比特流后直接下板验证都是行之有效的调试手段，合理配套使用能够极大地帮助自己寻找问题所在。在本次实验中，VAVIDO 的仿真波形功能就十分有效，想要具体了解模块内部运行细节，比如 Regfile、Hi、Lo 等寄存器数据的变化，可以在不影响模块功能的情况下，将一些内部信号也实时显示，这样便于直观观察，尽快找出问题所在。

然后，在硬件程序设计方面，也需要注意代码规范，宏定义的使用会方便程序的修改与移植。在《自己动手写 CPU》这本书中，使用了不少宏定义，这使得修改一些重要参数（如指令存储器的起始地址、指令的标号）等都十分方便。

最后的感想就是硬件设计不能一蹴而就，检验耗费的时间、精力都是软件测试不能比的，所以需要更多地耐心与真诚。本次实验也有一些不足与缺陷，因此后续的移植操作系统、编写可视化界面等实验，我会更加认真地去完成。