同濟大學
TONGJI UNIVERSITY

计算机系统实验课程实验报告

实验题目：MIPS 指令集流水线 CPU 改造

学号：2151769

姓名：吕博文

指导教师：郭玉臣

日期：2024.4.6

一、 实验环境部署与硬件配置说明

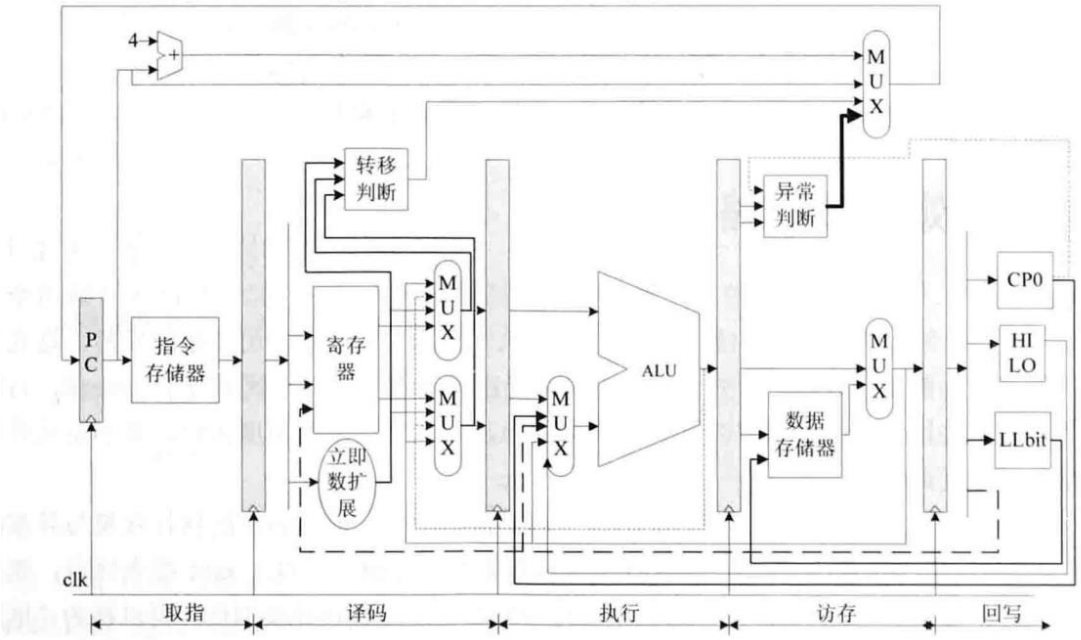
本次实验使用的实验环境是 Vivado 软件，Modelsim 软件，具体配置以及建立项目编写文件的格式和上学期完成流水线 CPU 时的部署配置基本一致，具体实验编写代码采用 Vivado 作为主项目编译器，利用 Vivado 自带的仿真功能和 ModelSim 的仿真功能进行前仿真，在下板方面，主板使用学校统一配置的 NEXYS-4 进行下板实验，汇编代码生成 COE 文件采用 MARS 模拟器进行实现，整个动态流水线项目涉及基本 Verilog 文件（.v 后缀）的编写，IP 核（coe 文件的导入）,前仿真以及后仿真的实现。

二、 实验的总体结构

实验要求：改造 54 条 CPU，支持到 89 条，实现 CP0,异常处理，可以参考《自己动手写 CPU》的 源代码，加以改造以后形成适配于 NEXYS-4 板的集成 CPU 模块，检查方式为：通过给定测试程序的 coe 文件（内部由 11 个测试函数构成，测试内容覆盖了 89 条 MIPS 指令，CP0，以及时钟中断）；最后下板需要达到目标：数码管低半字显示 0x0001，高半字随着时钟中断而计数。

1、OpenMIPSCPU 整体架构

本次 CPU 改造实验参考《自己动手写 CPU》一书前 11 章的内容，架构出包含异常处理，CP0 指令以及一系列 MIPS 常见指令的一个完善的 CPU 框架，其中 CPU 内部运行依然按照五步流水线的方式运行，整体 CPU 架构如下：



三、 实验过程与方法

3.1 实验过程

1、由于《自己动手写 CPU》一书中给出了参考代码，我们选择在其基础上进行修改以适应 N4 开发板，首先新建 vivado 工程文件，将 Chap11 中的所

有代码进行导入,其中包括头文件部分，CPU 主体部分，仿真文件部分；

2、为方便之后的在线仿真和下板实验，我单独定义了一个顶层模块 openmips_min_spoc_board.v 文件用来作为下板的顶层文件，在这个文件中，我也额外集成了分频器 divider.v,数码管模块 seg7x16.v 文件用以使得开发板上显示的结果更加清晰明了，同时约束文件 cpu84_contraints.xdc 文件是结合 N4 开发板具体的接口定义的。

3、导入 IP 核，我们将给定的测试程序 mips_89_mars_board_big.s 经过 MARS 软件生成对应的十六进制文件，在开头额外加入 memory_initialization_radix = 16;memory_initialization_vector = 两句话之后形成对应的测试 COE 文件，按照之前的方法将 COE 文件导入 Vivado 的 IP 核中形成具有数据存储的 IMEM 模块

4、具体修改部分:为适应 N4 开发板以及测试文件的具体要求，CPU 需要采用大端模式编写（原书中代码就是采用的这种方法），同时要求中断例程起始地址为 0x00400004，数码管显示地址为 0x10010000 的 DMEM 单元，为此我们额外定义了一些宏定义：

```
//额外添加的宏定义
`define PcBegin 32'h00400000
`define DataBegin 32'h10010000
`define ExceptionBegin 32'h00400004
```

针对 PcBegin，我们更改 PC reg 中的内容；
针对 DataBegin ，我们修改 data_ramz 中的内容，使得数据起始地址和 MARS 的标准保持一致；
针对 ExceptionBegin，我们修改 cp0_reg 中的内容，更改中断例程起始地址

5、仿真文件的更改：为方便观察仿真图形的结果，我们分别将 PC 值，inst 值和最后的 result 值实时传入顶层模块，通过仿真图形来观察结果，因为测试文件耗时较长，为了有效的观察到所有测试文件运行完之后的时钟中断结果，我们将 clk 设置为每 0.1ns 一更新

6、最后将 openmips_min_spoc_board.v 作为顶层模块进行下板测试，观察 N4 板上的结果是否符合测试要求的结果。

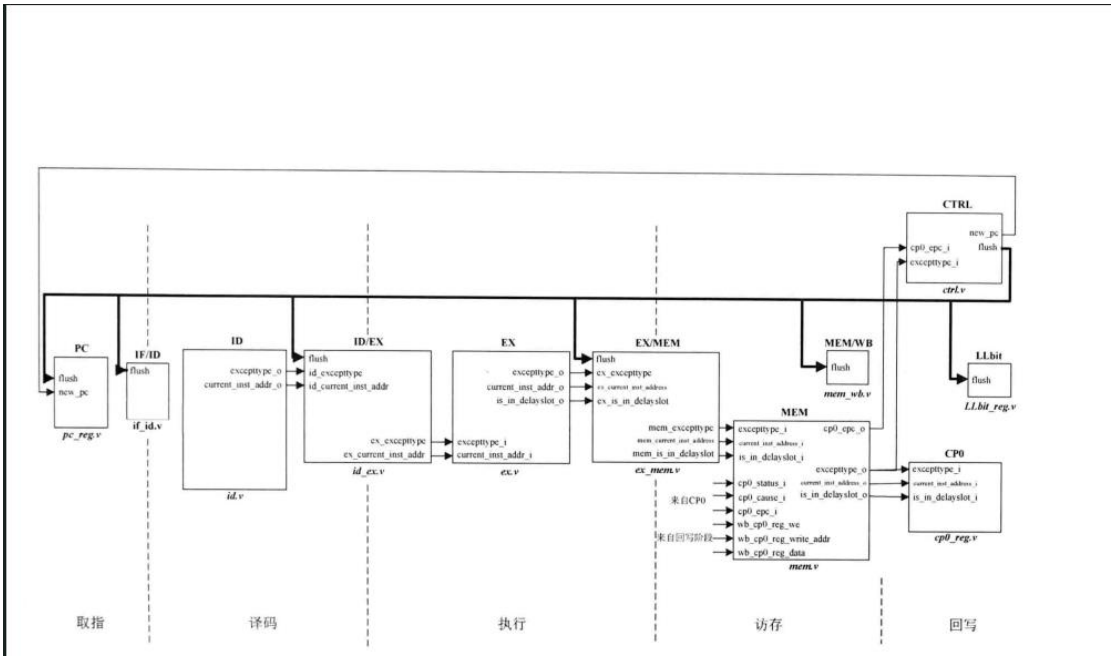
3.2 实验方法

本次实验测试方法主要以给定的测试文件测试为主，测试文件对于错误会打印并执行死循环，可以通过观察结果来判定错误出处进而改正错误；

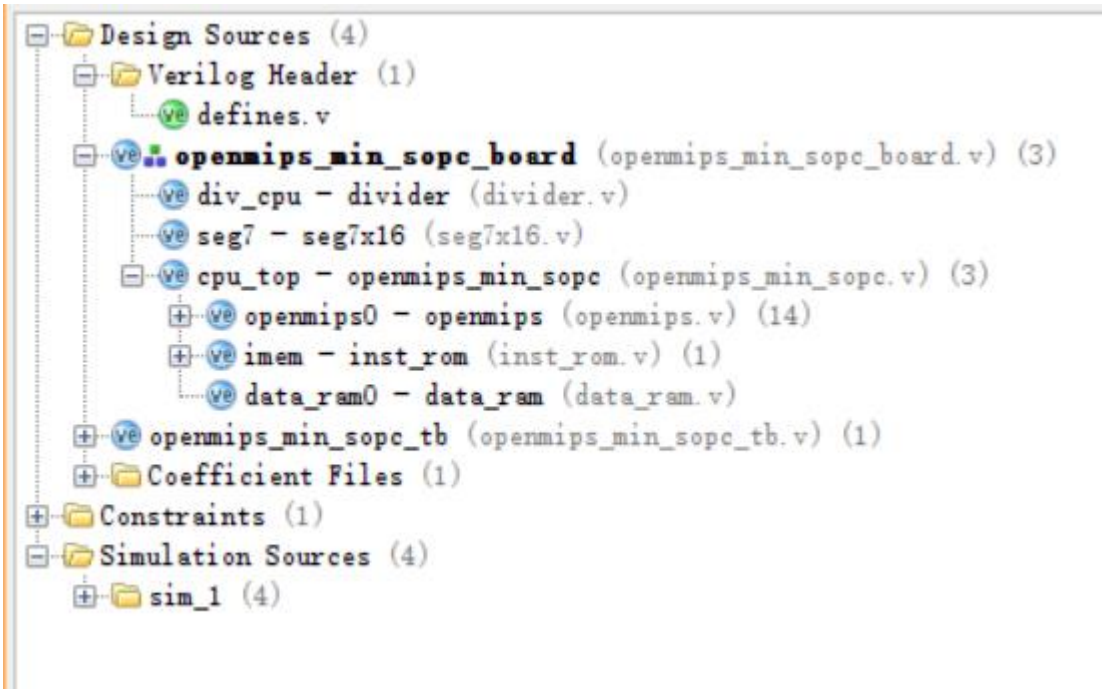
对于仿真，我们可以将需要查看的寄存器拖入波形图，再次执行 Relaunch Simulation 指令就可以查看对应的中间变量，进而改正错误。

四、 总体架构部件的解释说明

OpenMips CPU 整体模块：



项目结构如下：



下面分别对几个重要模块进行详细解释说明：

（1） 下板顶层模块

项目顶层模块，负责调用 CPU 模块进行 CPU 的运行，负责选择结果输出，分频以及调用七段数码管模块显示运算值。

```
`timescale 1ns / 1ps



module openmips_min_sopc_board(
    input clk_in,
    input reset,
```

```
    input [1:0]res_choose,
    output [7:0] o_seg,
    output [7:0] o_sel
    );
    wire clk_new;
    wire [31:0]seg_data;
    wire [31:0] inst,pc,result;

    divider#(100000)div_cpu(clk_in,reset,clk_new);

    seg7x16 seg7 (clk_in,reset,1,seg_data,o_seg,o_sel);
    assign seg_data = (res_choose[1]?inst:(res_choose[0]?pc:result));

    openmips_min_sopc cpu_top(clk_new,reset,pc,inst,result);

endmodule
```

（2） CPU 运算顶层模块

负责调用数据存储器模块，指令存储器模块和控制器模块

```
//模拟(simulation)顶层模块
module openmips_min_sopc(

    input wire  clk,
    input wire  rst,
    output [31:0] pc,
    output [31:0] inst,
    output [31:0] output_res
);

  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;

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

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

        .rom_addr_o(pc),
        .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)

    );

    wire [31:0] mips_pc ;
    assign mips_pc =  pc - `PcBegin;

    //指令存储器
    inst_rom imem(mips_pc[12:2],inst);

    //数据存储器
    data_ram data_ram0(
        .clk(clk),
        .ce(mem_ce_i),
        .we(mem_we_i),
        .addr_in(mem_addr_i),
        .sel(mem_sel_i),
        .data_i(mem_data_i),
        .data_o(mem_data_o),
        .result(output_res)
    );
endmodule
```

（3） PCReg 模块

这里相比源代码做出更改，起始地址设置为 `PcBegin`

```
`include "defines.v"
```

```
module pc_reg(

    input   wire                                    clk,
    input wire                                  rst,

    input wire[5:0]             stall,
    input wire                  flush,
    input wire[`RegBus]          new_pc,

    input wire                  branch_flag_i,
    input wire[`RegBus]          branch_target_address_i,

    output reg[`InstAddrBus]           pc,
    output reg                  ce

);

    always @ (posedge clk) begin
        if (ce == `ChipDisable) begin
            pc <= `PcBegin;
        end
        else if(ce != `ChipDisable)begin
            if(flush == 1'b1) begin
                pc <= new_pc;
            end else if(stall[0] == `NoStop) begin
                if(branch_flag_i == `Branch) begin
                    pc <= branch_target_address_i;
                end else begin
                pc <= pc + 4'h4;
            end
            end
        end
    end

    always @ (posedge clk) begin
        if (rst == `RstEnable) begin
            ce <= `ChipDisable;
        end else begin
            ce <= `ChipEnable;
        end
    end

endmodule
```

（4） Data_ram 模块

这里做出更改，数据起始地址改为 DataBegin

```
`include "defines.v"

module data_ram(

    input wire  clk,
    input wire  ce,
    input wire we,
    input wire[`DataAddrBus] addr_in,
    input wire[3:0]  sel,
    input wire[`DataBus] data_i,
    output reg[`DataBus]    data_o,
    output wire[`DataBus] result
);

    reg[`ByteWidth]  data_mem0[0:`DataMemNum-1];
    reg[`ByteWidth]  data_mem1[0:`DataMemNum-1];
    reg[`ByteWidth]  data_mem2[0:`DataMemNum-1];
    reg[`ByteWidth]  data_mem3[0:`DataMemNum-1];
    wire [`DataAddrBus] addr;
    // change part
    assign addr = addr_in - `DataBegin;
    assign result =
{data_mem3[0],data_mem2[0],data_mem1[0],data_mem0[0]};
    //assign result = 1;
    always @ (posedge clk) begin
        if (ce == `ChipDisable) begin
            //data_o <= ZeroWord;
        end else if(we == `WriteEnable) begin
              if (sel[3] == 1'b1) begin
              data_mem3[addr[`DataMemNumLog2+1:2]] <= data_i[31:24];
            end
              if (sel[2] == 1'b1) begin
              data_mem2[addr[`DataMemNumLog2+1:2]] <= data_i[23:16];
            end
            if (sel[1] == 1'b1) begin
              data_mem1[addr[`DataMemNumLog2+1:2]] <= data_i[15:8];
            end
              if (sel[0] == 1'b1) begin
              data_mem0[addr[`DataMemNumLog2+1:2]] <= data_i[7:0];
            end
        end
    end
```

```
    always @ (*) begin
        if (ce == `ChipDisable) begin
            data_o <= `ZeroWord;
      end else if(we == `WriteDisable) begin
            data_o <= {data_mem3[addr[`DataMemNumLog2+1:2]],
                       data_mem2[addr[`DataMemNumLog2+1:2]],
                       data_mem1[addr[`DataMemNumLog2+1:2]],
                       data_mem0[addr[`DataMemNumLog2+1:2]]};
        end else begin
                data_o <= `ZeroWord;
        end
    end

endmodule
```

（5） Ctrl 模块

这里做出更改，陷入中断以后，更改中断例程首地址为 ExceptionBegin

```
`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,

    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 <= `ExceptionBegin;
                  end
                  32'h00000008:       begin   //syscall
                      new_pc <= `ExceptionBegin;
                  end
                  32'h0000000a:       begin   //inst_invalid
                      new_pc <= `ExceptionBegin;
                  end
                  32'h0000000d:       begin   //trap
                      new_pc <= `ExceptionBegin;
                  end
                  32'h0000000c:       begin   //ov
                      new_pc <= `ExceptionBegin;
                  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
```

（6） **cpu84_constraint.xdc**

项目约束文件，根据 N4 开发板的具体接口编写：

```
create_clock -period 100.000 -name clk_pin -waveform {0.000 50.000}
[get_ports clk_in]
set_input_delay -clock [get_clocks *] 1.000 [get_ports reset]
```

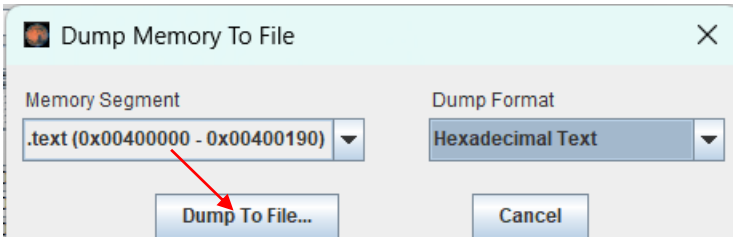
```
set_output_delay -clock [get_clocks *] 0.000 [get_ports -filter { NAME
=~  "*" && DIRECTION == "OUT" }]

set_property PACKAGE_PIN E3 [get_ports clk_in]
set_property PACKAGE_PIN N17 [get_ports reset]

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 PACKAGE_PIN J15 [get_ports {res_choose[0]}]
set_property PACKAGE_PIN L16 [get_ports {res_choose[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {res_choose[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {res_choose[1]}]

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 reset]
set_property IOSTANDARD LVCMOS33 [get_ports clk_in]
```

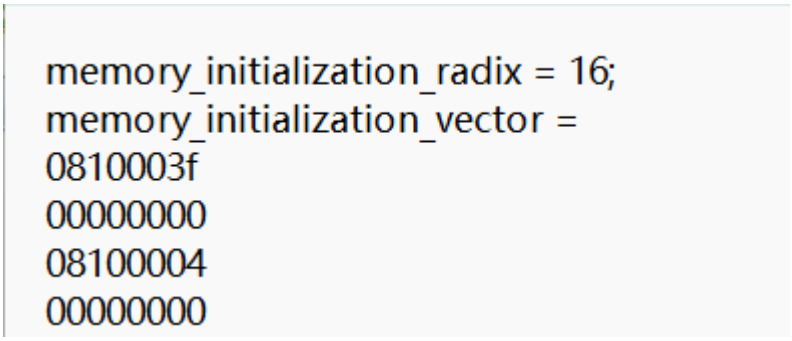
五、 实验仿真过程

进行前仿真测试，首先我们需要根据对应的测试要求完成汇编语言程序的编写。

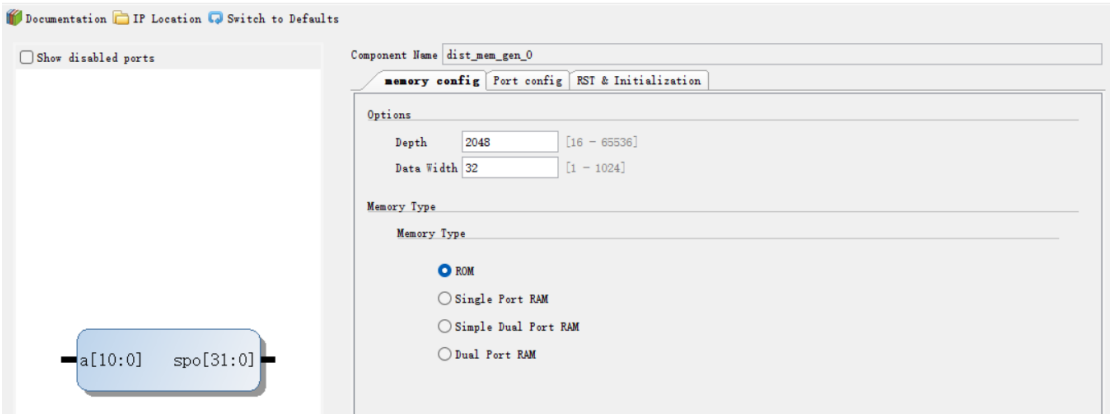
之后我们利用 MARS 进行 COE 文件的导出



导出的 COE 文件加入前两行之后即可作为 IP 核的内容导入：



之后，我们按要求配置好 Vivado 的 IP 核



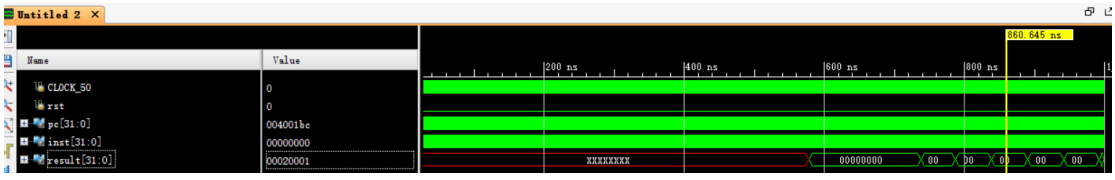
我们写出 testbench 测试程序如下：

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

module openmips_min_sopc_tb();
```

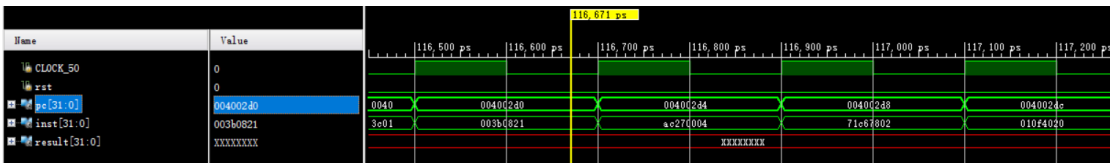
```
  reg     CLOCK_50;
  reg     rst;
  wire [31:0] pc;
  wire [31:0] inst;
  wire [31:0] result;

  initial begin
    CLOCK_50 = 1'b0;
    forever #0.1 CLOCK_50 = ~CLOCK_50;
  end

  initial begin
    rst = `RstEnable;
    #20 rst= `RstDisable;
    #10000 $stop;
  end

  openmips_min_sopc openmips_min_sopc0(
    .clk(CLOCK_50),
    .rst(rst),
    .pc(pc),
    .inst(inst),
    .output_res(result)
  );

endmodule
```

仿真过程

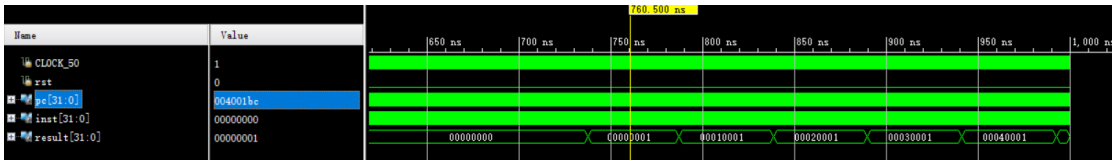
得到如下波形图：



通过上述整体波形图我们可以看出一开始，正在执行测试代码，pc 和 inst 值一直不断变化，11 条测试指令执行完之后进入时钟中断阶段，result 值开始有意义，具体而言：



通过放大波形图可以看到 PC 值和 inst 的值变化过程。

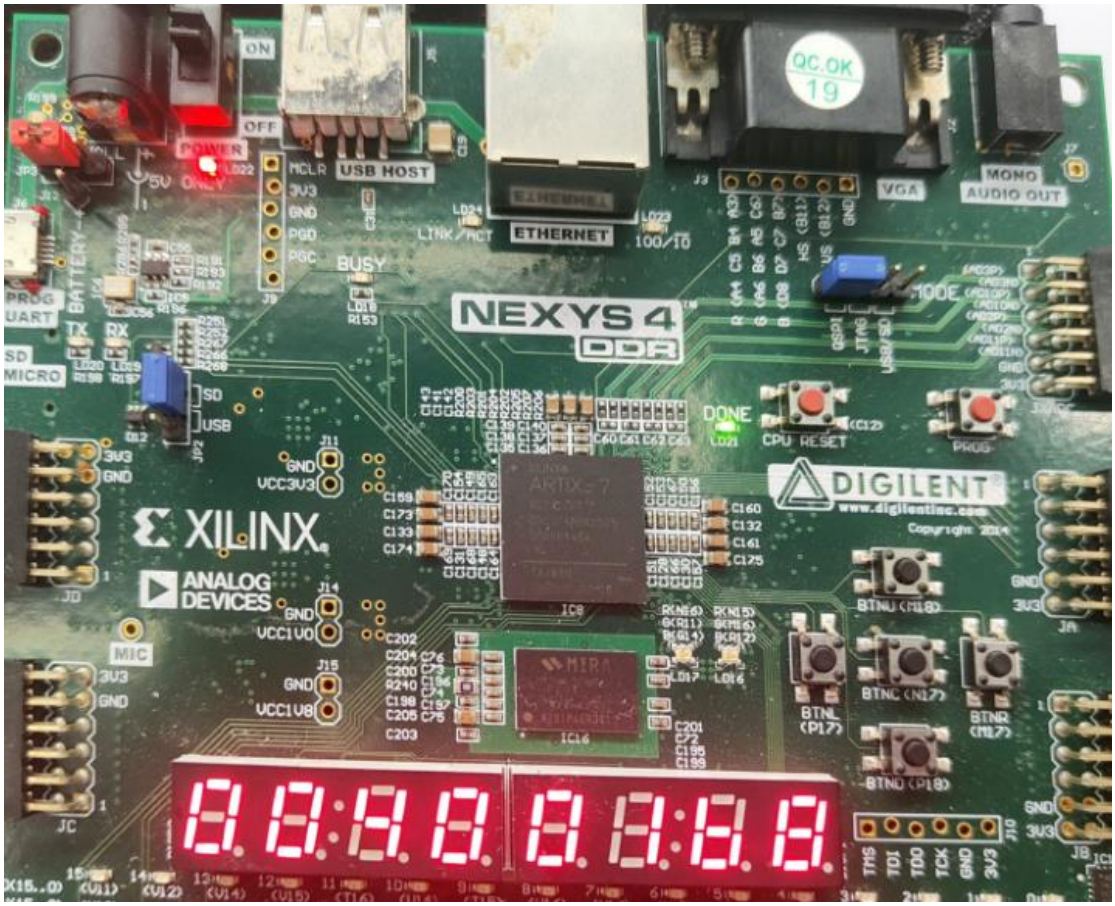


通过观察仿真后半部分，可以观察到 result 低四位始终为 1，高四位随时钟中断不断自增加一，符合测试程序的要求。

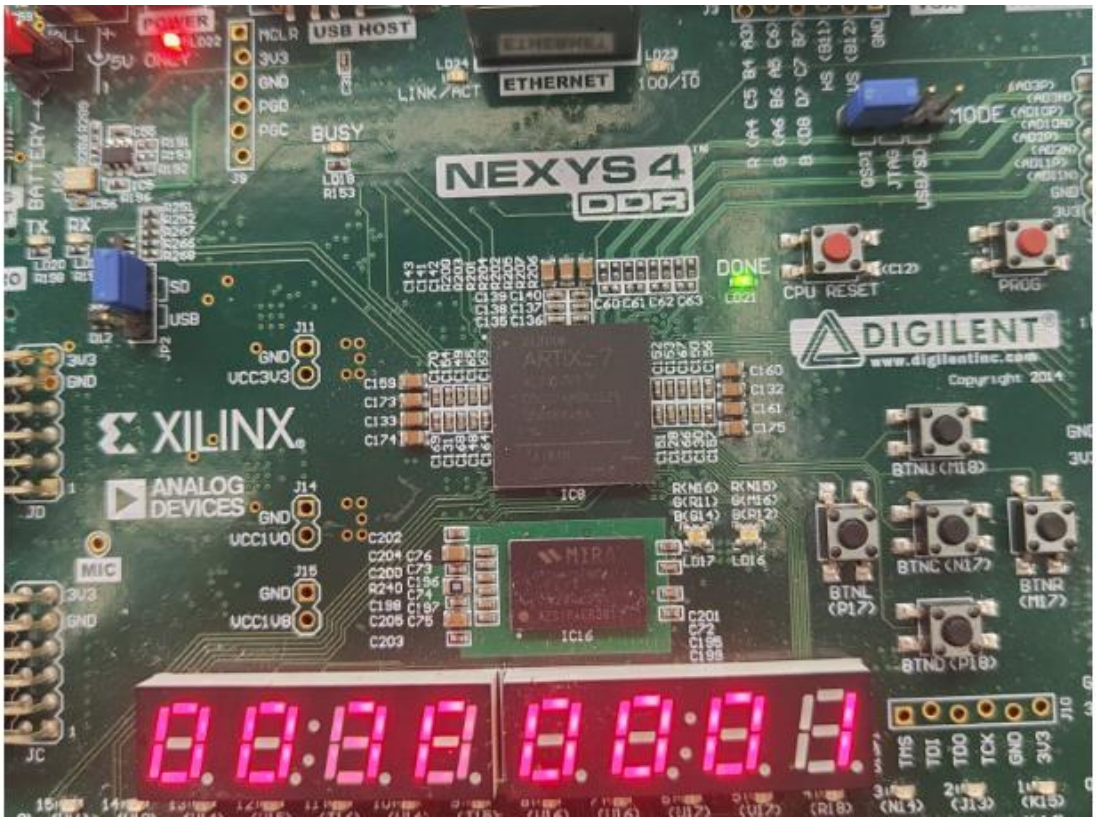
六、 实验验算程序下板测试过程与实现

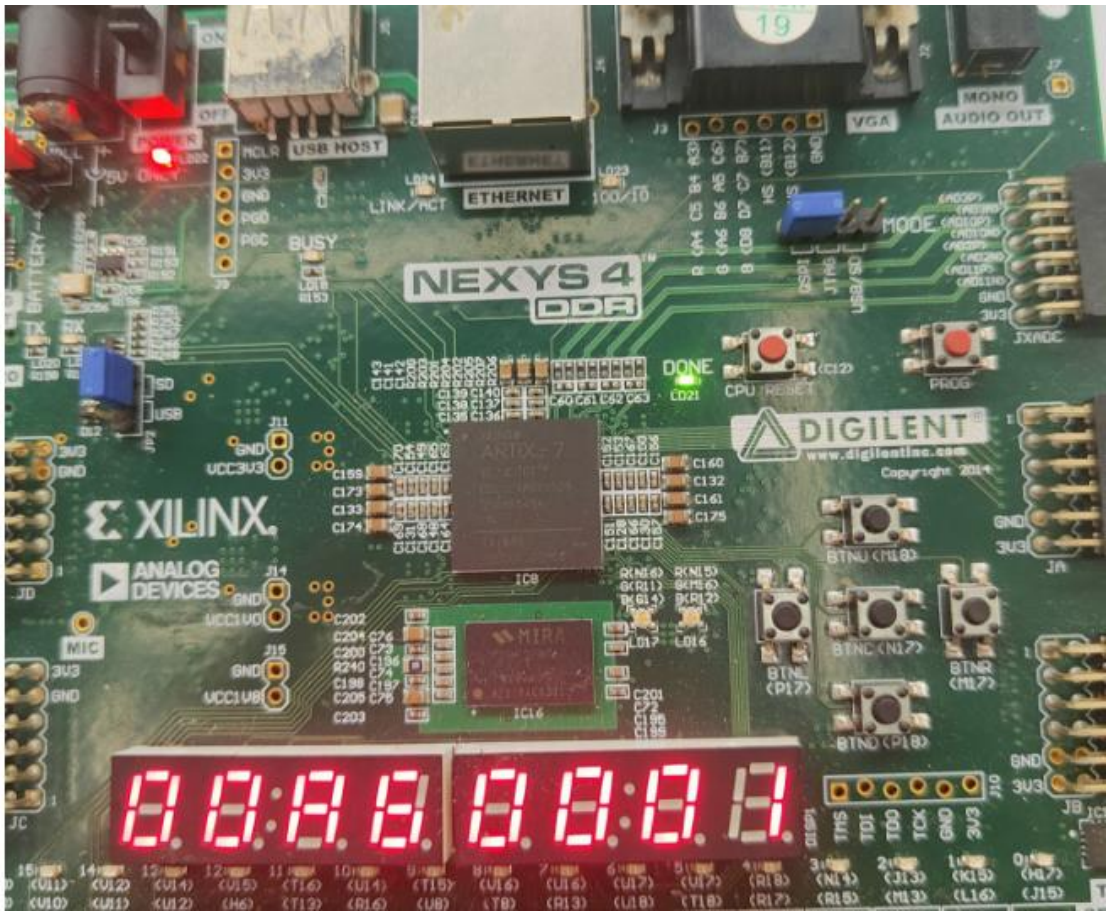
下板测试时需要注意的是首先要编写好对应的 xdc 文件，要对 CPU 模块传入的时钟信号进行分频方便人眼识别。

下板初期 PC 值：



时钟中断阶段 result 值变化：


NEXYS 4
DDR
DIGILENT
www.digilentinc.com
XILINX
ANALOG DEVICES
ETHERNET
USB HOST
CPU RESET
PROG
BTNU (M18)
BTNL (P17)
BTNC (N17)
BTNR (M17)
BTND (P18)
MODE
BUSY
DONE


NEXYS 4
DDR
DIGILENT
www.digilentinc.com
XILINX
ANALOG DEVICES
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
CPU RESET
PROG
BTNU (M18)
BTNL (P17)
BTNC (N17)
BTNR (M17)
BTND (P18)
MODE
BUSY
DONE

七、 总结与体会

本次 MIPS CPU 改造实验是整个计算机系统实验课程设计的第一阶段，经过对于《自己动手写 CPU》这本书中 CPU 结构的了解和实践，我对于整个流水线 CPU 架构有了更深刻的理解，同时我注意到本书最后给出的源码具有高度集成模块化的特定，在支持 89 条指令的前提下整个 CPU 模块代码数基本 1000 左右，相比我上学期实现的流水线 CPU 有着极大的可读性和扩展性，这一点值得我持续学习。

通过本次实验，我觉得值得注意的是学习到了一些 debug 的小技巧，对于硬件程序，波形图仿真验证，比特流下板验证都是有效的调试手段，在本次实验中，我主要使用了 Vivado 的波形仿真验证功能，可以查看我想看到的寄存器数据，方便了我调试模块，改正错误。

经过了 CPU 部分的改造，紧接着的就是 CPU 扩展到总线以及操作系统的移植，这一部分需要更多的学习时间去了解 CPU 在整个运行过程中的地位，了解我们需要扩展的总线结构，了解 Vivado 如何添加外接的 Flash，串口程序，如何最后实现 uc/os 操作系统的移植，接下来我将首先学习这部分的一些知识，参考《自己动手写 CPU》的进阶部分，完成接下来的这个实验。