计算机系统结构课程设计实验报告

——MIPS 指令集流水线 CPU 改造

院　　系　　电子与信息工程学院

专　　业　　计算机科学与技术

姓　　名

学　　号

题　　目　　89 条指令 CPU 改造

指导老师　　郭玉臣

完成日期　　2024.3.29

一、实验环境与实验内容

1.1 实验环境与硬件配置

处理器： 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz 3.11 GHz

开发平台：vivado 2019.2、ModelSim PE 10.4c

测试环境：MARS4.5

开发板： NEXYS 4 DDR Atrix-7

1.2 实验内容

1.2.1 实验目的

1. 回顾 MIPS 五阶段流水线 CPU 的基本结构
2. 学习教学用 CPU 各模块的编写以及模块之间的连接结构
3. 加深对 Verilog 语言的理解以及对硬件开发的理解
4. 为之后移植操作系统做准备工作

1.2.2 实验方案

参考《自己动手写 CPU》，改造 54 条指令 MIPS 流水线 CPU，使其支持 89 条指令，实现 CP0、异常处理。在《自己动手写 CPU》源码的基础上增加实现指令功能，完善总线及控制器等实现。本实验需要新增的 35 条指令具体如下所示：

1. 移动指令指令：movn，movz
2. 算术指令指令：clo，madd，maddu，msub，msubu
3. 跳转指令指令：b，bal，bgezal，bgtz，blez，bltz，bltzal
4. 加载存储指令指令：ll，lwl，lwr，sc，swl，swr
5. 异常相关指令指令：tge,tgeu,tlt,tltu,tne,teqi,tgei,tgeiu,tlti,tltiu,tnei
6. 其他指令：nop，ssnop，sync，pref

本实验中实现的 MIPS CPU 为五级流水线 CPU， 分别又取指、译码、执行、访存、回写五个阶段，存储模式采取大端模式，整体采用哈佛结构，有独立的指令存储器和数据存储器。

二、指令与数据通路说明

2.1 指令说明

1. movn
    1. 格式：movn rd,rs,rt
    2. 描述：如果 rt 不等于 0，则 rd 获取 rs 的值。如果 rt 等于 0，则 rd 的值不变，movn 指令的功能是在非零条件下移动寄存器的值。
    3. 涉及的硬件：PC（程序计数器）、NPC（下一条指令的程序计数器）、IMEM（指令存储器）、RegFile（寄存器文件）。
2. movz
    1. 格式：movz rd,rs,rt
    2. 描述：如果 rt 等于 0，则 rd 获取 rs 的值。与 movn 指令相对，其功能是在零条件下移动寄存器的值。
    3. 涉及的硬件：PC、NPC、IMEM、RegFile。
3. clo
    1. 格式：clo rd,rs
    2. 描述：该指令用于计算 rs 寄存器中最高位开始连续的“1”的个数。例如，如果 rs 寄存器的值为 0xFFFFFFFF，则 clo 指令会计算出 32，因为该数值的二进制表示是连续 32 个“1”。
    3. 涉及的硬件：PC、NPC、IMEM、RegFile。
4. madd
    1. 格式：madd rs,rt
    2. 描述：该指令用于执行乘法加法操作，它将 rs 和 rt 寄存器的值相乘，然后将结果加到特殊寄存器 HI,LO 中。如果乘法结果超出了 32 位，HI 寄存器存储高 32 位，而 LO 寄存器存储低 32 位。
    3. 涉及的硬件：PC、NPC、IMEM、RegFile、HI,LO。
5. maddu
    1. 格式：maddu rs,rt
    2. 描述：该指令类似于 madd，但它用于无符号数的乘法加法操作。同样地，乘法结果的高位存储在 HI 寄存器，低位存储在 LO 寄存器。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、HI,LO。

6. msub

1. 格式：msub rs,rt

2. 描述：此指令将 rs 和 rt 寄存器的值相乘，然后将结果从 HI,LO 寄存器中的现有值中减去。如果乘法结果超出了 32 位，则 HI 寄存器存储高 32 位，而 LO 寄存器存储低 32 位。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、HI,LO。

7. msubu

1. 格式：msubu rs,rt

2. 描述：当执行 rs 和 rt 的无符号数乘法操作，结果会从特殊寄存器 HI,LO 中现有的值中减去。如果 rs 和 rt 的乘法结果超过了 32 位，高位存储在 HI 寄存器，低位存储在 LO 寄存器。msubu 指令的作用是执行无符号数乘法后的减法操作。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、HI,LO。

8. b

1. 格式：b offset

2. 描述：无条件分支，当指令执行时，处理器会跳转到 b 指令的当前位置加上 offset 的目标地址。b 指令简化了代码流程控制，可以实现无条件的跳转。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

9. bal

1. 格式：bal offset

2. 描述：无条件分支，同时将返回地址存储在寄存器 $ra 中，方便函数调用后返回到原程序继续执行。bal 指令等效于 bgezal 指令的特殊情况，即 bgezal 的条件总是满足的情况，因此总是执行跳转。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

10. bgezal

1. 格式：bgezal rs,offset

2. 描述：如果 rs 大于等于 0，则执行分支，并且将返回地址保存在寄存器 $ra 中，用于支持函数的调用与返回。该指令在实现函数调用时很有用，因为它提供了一个链接到返回地址的机制。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

11. bgtz

1. 格式：bgtz rs,offset
2. 描述：如果 rs 大于 0，则执行分支操作。bgtz 指令用于条件分支，通常与循环或条件语句结合使用，以实现更复杂的控制流程。
3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

12. blez

1. 格式：blez rs,offset
2. 描述：如果 rs 小于等于 0，则执行分支操作，处理器跳转到 blez 指令的当前位置加上 offset 的目标地址。blez 指令用于在 rs 寄存器的值不大于零时控制程序流程。
3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

13. bltz

1. 格式：bltz rs,offset
2. 描述：如果 rs 小于 0，则执行分支操作，处理器跳转到 bltz 指令的当前位置加上 offset 的目标地址。bltz 指令专用于在 rs 寄存器的值为负数时实现程序的条件分支。
3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

14. bltzal

1. 格式：bltzal rs,offset
2. 描述：如果 rs 小于 0，则执行分支，并且将返回地址保存在寄存器 $ra 中。bltzal 指令不仅用于条件分支，而且用于支持函数调用与返回，使得程序能够在执行完函数后返回到原来的位置。
3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT18、ADD。

15. ll

1. 格式：ll rt, offset(base)
2. 描述：ll 指令用于从内存中加载一个字到寄存器 rt，同时设置内存系统的一个监测位，用于后续的 sc（Store Conditional）指令实现原子操作。如果在执行 sc 指令之前该地址被修改，sc 操作将失败。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、DMEM。

16. `lwl`

1. 格式：`lwl rt, offset(base)`

2. 描述：该指令用于加载字的左边部分到寄存器 `rt`，一般与 `lwr` 指令组合使用来加载非对齐的字。`lwl` 指令根据对齐的字的边界，从内存中读取字的左边部分，拼接到寄存器 `rt` 中。与 `lhu`、`lh` 指令的作用不同，后两者是加载半字。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT16、DMEM。

17. `lwr`

1. 格式：`lwr rt, offset(base)`

2. 描述：与 `lwl` 指令相对应，用于加载字的右边部分到寄存器 `rt`，一般与 `lwl` 指令组合使用来加载非对齐的字。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT16、DMEM。

18. `sc`

1. 格式：`sc rt, offset(base)`

2. 描述：该指令用于实现原子交换操作，也就是 LLbit（Load Linked bit）为 1 时，将寄存器 `rt` 的内容存储到指定的内存地址，并将 LLbit 设为 0。如果 RMW（Read Modify Write）指令组执行了此操作，也就是 LLbit 为 0，则不会执行存储，并将寄存器 `rt` 中的内容设为 0。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT16、DMEM。

19. `swl`

1. 格式：`swl rt, offset(base)`

2. 描述：将寄存器 `rt` 中的字的左边部分存储到内存的指定地址中，通常与 `swr` 指令配合使用以实现非对齐的字的存储操作。`swl` 指令根据对齐的字边界，将寄存器 `rt` 中的左边部分存储到内存。与 `sh` 指令的作用不同，后者是存储半字。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT16、DMEM。

20. `swr`

1. 格式：`swr rt, offset(base)`

2. 描述：将寄存器 `rt` 中的字的右边部分存储到内存的指定地址中，通常与 `swl` 指令配合使用以实现非对齐的字的存储操作。

3. 涉及的硬件：PC、NPC、IMEM、RegFile、ALU、EXT16、DMEM。

21. `tge`

1. 格式：`tge rs, rt`
2. 描述：如果通用寄存器 `rs` 的值大于等于寄存器 `rt` 的值，则执行陷阱（trap）操作。`tge` 指令用于在寄存器的值满足特定的条件时产生陷阱，用于程序的异常处理或调试。
3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

22. `tgeu`

1. 格式：`tgeu rs, rt`
2. 描述：如果通用寄存器 `rs` 的值在无符号比较下大于等于寄存器 `rt` 的值，则执行陷阱操作。`tgeu` 与 `tge` 指令相似，但专用于无符号数的比较。
3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

23. `tlt`

1. 格式：`tlt rs, rt`
2. 描述：如果通用寄存器 `rs` 的值小于寄存器 `rt` 的值，则执行陷阱操作。`tlt` 指令用于在寄存器的值不满足特定的条件时产生陷阱，用于程序的异常处理或调试。
3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

24. `tltu`

1. 格式：`tltu rs, rt`
2. 描述：如果通用寄存器 `rs` 的值在无符号比较下小于寄存器 `rt` 的值，则执行陷阱操作。`tltu` 指令用于在寄存器的值不满足特定的无符号比较条件时产生陷阱，用于程序的异常处理或调试。
3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

25. `tne`

1. 格式：`tne rs, rt`
2. 描述：如果通用寄存器 `rs` 的值不等于寄存器 `rt` 的值，则执行陷阱操作。`tne` 指令用于在寄存器的值不满足等值条件时产生陷阱，用于程序的异常处理或调试。
3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

26. teqi

1. 格式：teqi rs, immediate

2. 描述：如果通用寄存器 rs 的值等于立即数（立即数被符号扩展到 32 位）则执行陷阱操作。teqi 指令用于比较寄存器值和立即数，如果相等，则产生陷阱。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

27. tgei

1. 格式：tgei rs, immediate

2. 描述：如果通用寄存器 rs 的值大于等于立即数（立即数被符号扩展到 32 位）则执行陷阱操作。tgei 指令用于比较寄存器值和立即数，如果寄存器值大等于立即数，则产生陷阱。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

28. tgeiu

1. 格式：tgeiu rs, immediate

2. 描述：如果通用寄存器 rs 的值在无符号比较下大于或等于立即数 immediate，则执行陷阱操作。tgeiu 指令用于在寄存器的值无符号比较满足或超过特定立即数时产生陷阱，通常用于程序的异常处理或调试。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

29. tlti

1. 格式：tlti rs, immediate

2. 描述：如果通用寄存器 rs 的值小于立即数 immediate，则执行陷阱操作。tlti 指令用于在寄存器的值小于特定立即数时产生陷阱，用于程序的异常处理或调试。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

30. tltiu

1. 格式：tltiu rs, immediate

2. 描述：如果通用寄存器 rs 的值在无符号比较下小于立即数 immediate，则执行陷阱操作。tltiu 指令用于在寄存器的值无符号比较小于特定立即数时产生陷阱，用于程序的异常处理或调试。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

31. tnei

1. 格式：tnei rs, immediate

2. 描述：如果通用寄存器 rs 的值不等于立即数 immediate，则执行陷阱操作。tnei 指令用于在寄存器的值不等于特定立即数时产生陷阱，用于程序的异常处理或调试。

3. 涉及的硬件：PC、NPC、CPO、IMEM、RegFiles、ALU。

32. nop

1. 描述：无操作。

33. ssnop

1. 描述：一个特殊类型的空操作，此类指令在某些 MIPS CPU 中的 nop 指令的使用上有特殊的效果，可以防止与 nop 指令相关的潜在执行速度降低的问题。

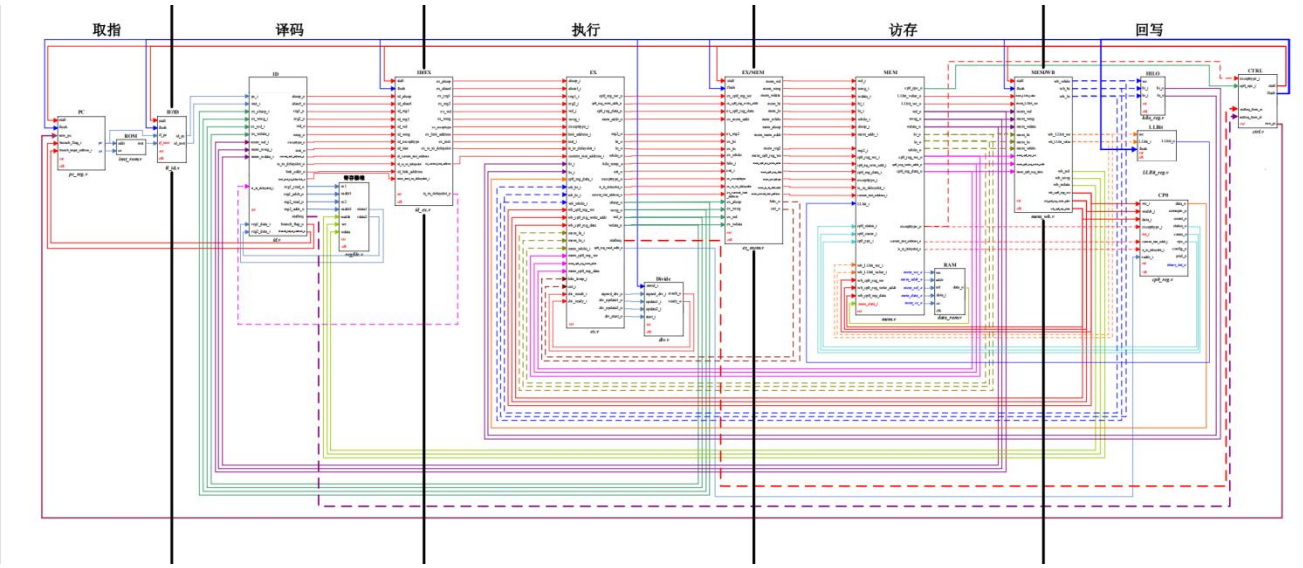
34. sync

1. 描述：用于保证内存操作的完成，以确保在此指令之前的所有内存操作都已经正确完成，以确保数据的一致性。在多核心或多线程环境中，sync 指令用于在不同的处理器或线程之间同步内存操作。

35. pref

1. 描述：用于数据预取操作，此指令在 MIPS CPU 中用于预先将数据加载到缓存中，可以提升后续访问数据的速度。pref 指令可视为 nop 指令的增强版。

2.2 整体数据通路



三、修改模块说明

3.1 顶层模块 openmips_min_sopc.v

本模块需要将 CPU 模块、指令存储器、数据存储器、七段数码管控制器实例化，并完成个模块之间的连接。同时顶层模块中需要实现分频器，保证数码管的显示变化可以被人眼捕捉。具体实现为：

```
1.  `include "defines.v"
2.
3.  module openmips_min_sopc(
4.          input wire clk,
5.          input wire rst,
6.          output [7:0] O_Seg,
7.          output [7:0] O_Sel
8.  );
9.
10.   reg clk_s;
11.   initial clk_s=0;
12.   integer cnt=0;
13.   always @(posedge clk)
14.   begin
15.
16. if(cnt<1)
17.     cnt=cnt+1;
18.   else
19.      begin
20.         cnt=0;
21.         clk_s=~clk_s;
22.      end
23.   end
24.
25.
26.   //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);
```

```
78.  endmodule
```

3.2 控制器模块 ctrl.v

中断例程起始地址为 0x00400004，需要修改 ctrl.v 中的跳转地址，将中断跳转地址修改为 0x00400004，从而保证最后输出结果符合预期。具体实现为：

```
1.  `include "defines.v"
2.
3.  module ctrl(
4.   input wire rst,
5.   input wire[31:0] excepttype_i,
6.   input wire[`RegBus] cp0_epc_i,
7.   input wire stallreq_from_id,
8.    //来自执行阶段的暂停请求
9.   input wire stallreq_from_ex,
10.  output reg[`RegBus] new_pc,
11.  output reg flush,
12.  output reg[5:0] stall
13. );
14.  always @ (*) begin
15.   if(rst == `RstEnable) begin
16.    stall <= 6'b000000;
17.    flush <= 1'b0;
18.    new_pc <= `ZeroWord;
19.   end else if(excepttype_i != `ZeroWord) begin
20.    flush <= 1'b1;
21.    stall <= 6'b000000;
22.    case (excepttype_i)
23.     32'h00000001:  begin   //interrupt
24.      new_pc <= 32'h00000020;
25.     end
26.     32'h00000008:  begin   //syscall
27.      new_pc <= 32'h00400004;
28.     end
29.     32'h0000000a:  begin   //inst_invalid
30.      new_pc <= 32'h00400004;
31.     end
32.     32'h0000000d:  begin   //trap
33.      new_pc <= 32'h00400004;
34.     end
35.     32'h0000000c:  begin   //ov
36.      new_pc <= 32'h00400004;
37.     end
38.     32'h0000000e:  begin   //eret
39.      new_pc <= cp0_epc_i;
40.     end
41.     default : begin
42.     end
43.    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 DMEM 模块 data_ram.v

在 DMEM 中，需要减去起始地址 0x00400000 进行地址变换

```
`include "defines.v"

module data_ram(
 input wire          clk,
 input wire          ce,
 input wire          we,
 input wire[`DataAddrBus]   addr,
 input wire[3:0]        sel,
 input wire[`DataBus]      data_i,
 output reg[`DataBus]     data_o,
 output[`DataBus] debugreg

);
 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];


 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]-
17'b0_0100_0000_0000_0000] <= data_i[31:24];
      end
     if (sel[2] == 1'b1) begin
        data_mem2[addr[`DataMemNumLog2+1:2]-
17'b0_0100_0000_0000_0000] <= data_i[23:16];
      end
```

```
        if (sel[1] == 1'b1) begin
          data_mem1[addr[`DataMemNumLog2+1:2]-17'b0_0100_0000_0000_0000] <= data_i[15:8];
        end
       if (sel[0] == 1'b1) begin
          data_mem0[addr[`DataMemNumLog2+1:2]-17'b0_0100_0000_0000_0000] <= 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]-17'b0_0100_0000_0000_0000],
                  data_mem2[addr[`DataMemNumLog2+1:2]-17'b0_0100_0000_0000_0000],
                  data_mem1[addr[`DataMemNumLog2+1:2]-17'b0_0100_0000_0000_0000],
                  data_mem0[addr[`DataMemNumLog2+1:2]-17'b0_0100_0000_0000_0000]};
    end else begin
      data_o <= `ZeroWord;
    end
   end

      assign debugreg={data_mem3[17'b0_0000_0000_0000_0000],
                       data_mem2[17'b0_0000_0000_0000_0000],
                       data_mem1[17'b0_0000_0000_0000_0000],
                       data_mem0[17'b0_0000_0000_0000_0000]};
endmodule
```

3.4 七段数码管模块 seg7x16.v

将指定的 0x10010000 内存单元最终结果显示在数码管上，实现沿用 MIPS54 的气短数码管驱动模块。具体实现为：

```
`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
```

3.5 约束文件修改

约束文件主要修改选择信号、时钟信号、复位信号引脚，具体实现修改为

```
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]}]
```

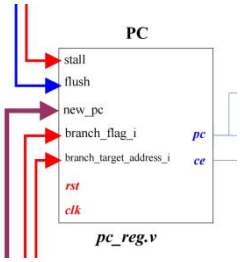
```
18.  set_property PACKAGE_PIN T14 [get_ports {O_Sel[5]}]
19.  set_property PACKAGE_PIN K2 [get_ports {O_Sel[6]}]
20.  set_property PACKAGE_PIN U13 [get_ports {O_Sel[7]}]
21.
22.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[7]}]
23.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[6]}]
24.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[5]}]
25.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[4]}]
26.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[3]}]
27.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[2]}]
28.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[1]}]
29.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Sel[0]}]
30.
31.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[7]}]
32.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[6]}]
33.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[5]}]
34.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[4]}]
35.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[3]}]
36.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[2]}]
37.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[1]}]
38.  set_property IOSTANDARD LVCMOS33 [get_ports {O_Seg[0]}]
39.
40.  set_property IOSTANDARD LVCMOS33 [get_ports rst]
41.  set_property IOSTANDARD LVCMOS33 [get_ports clk]
42.
43.  create_clock -period 200.000 -name clk_pin -waveform {0.000 100.000} [get_ports clk]
44.  set_input_delay -clock [get_clocks *] 1.000 [get_ports rst]
45.  set_output_delay -clock [get_clocks *] 0.000 [get_ports -
filter { NAME =~  "*" && DIRECTION == "OUT" }]
```

四、模块说明

4.1 PC 寄存器模块

功能：存储当前指令的地址，并根据控制信号和指令流水线的状态更新 PC 寄存器的值。

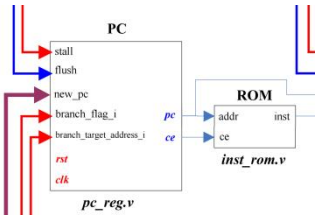
模块设计：



4.2 IF 模块

功能：取指模块，主要包含指令寄存器等。

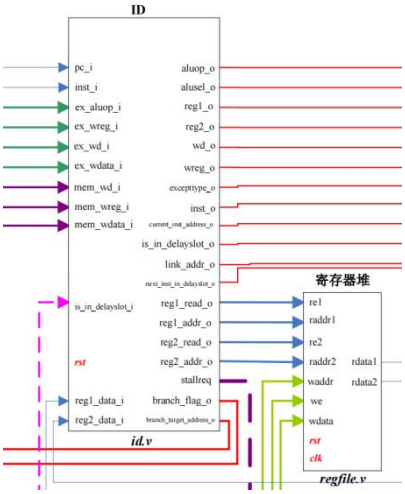
模块设计：



4.3 ID 模块

功能：主要包含控制单元模块、通用寄存器堆等

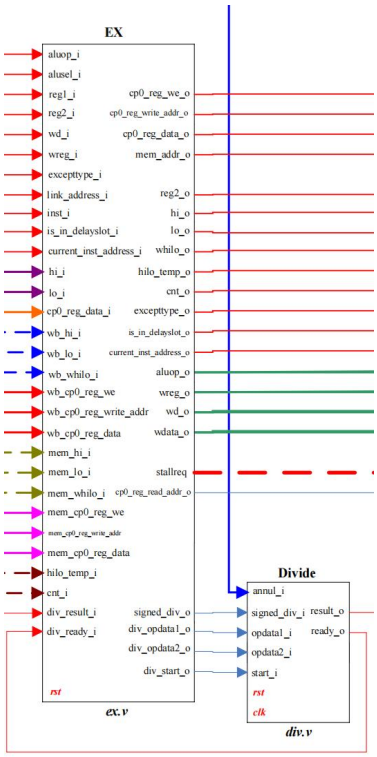
模块设计：



4.4 EXE 模块

功能：执行模块，包含除法器模块。

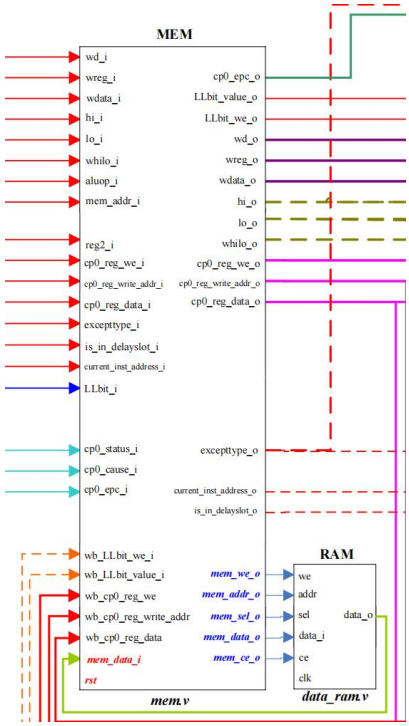
模块设计：



4.5 MEM 模块

功能：访存模块，主要包含数据存储器等模块

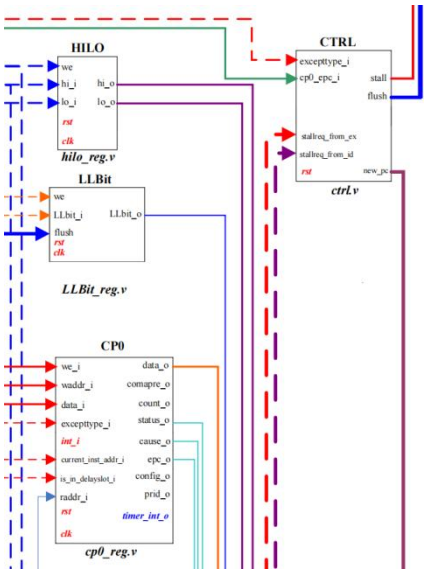
模块设计：



4.6 WB 模块

功能：回写模块，主要包含 HI 寄存器、LO 寄存器、CP0 寄存器等模块。

模块设计：



五、仿真与下板验证

5.1 测试与调试方法

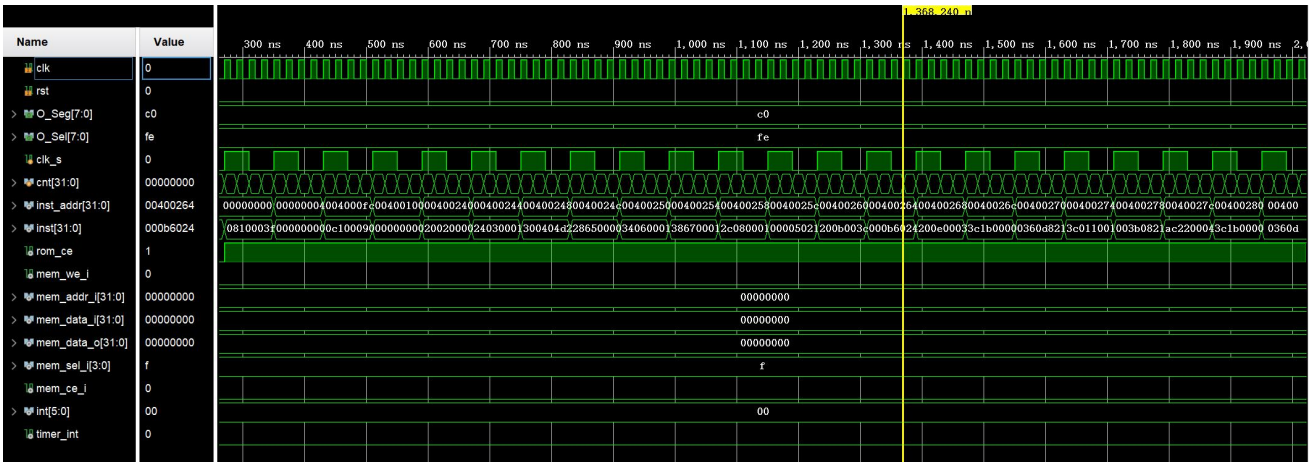
本次实验采用 Vivado 2019.2 自带仿真工具进行前后仿真，观察波形变化。采用的测试 testbench 文件实现如下：

```
1.  `include "defines.v"
2.  `timescale 1ns/1ps
3.
4.  module openmips_min_sopc_tb();
5.
6.    reg     CLOCK_50;
7.    reg     rst;
8.
9.
10.   initial begin
11.     CLOCK_50 = 1'b0;
12.     forever #10 CLOCK_50 = ~CLOCK_50;
13.   end
14.
15.   initial begin
16.     rst = `RstEnable;
17.   end
18.
19.   openmips_min_sopc openmips_min_sopc0(
20.   .clk(CLOCK_50),
21.   .rst(rst)
22.  );
23.
24. endmodule
```

5.2 仿真验证与分析

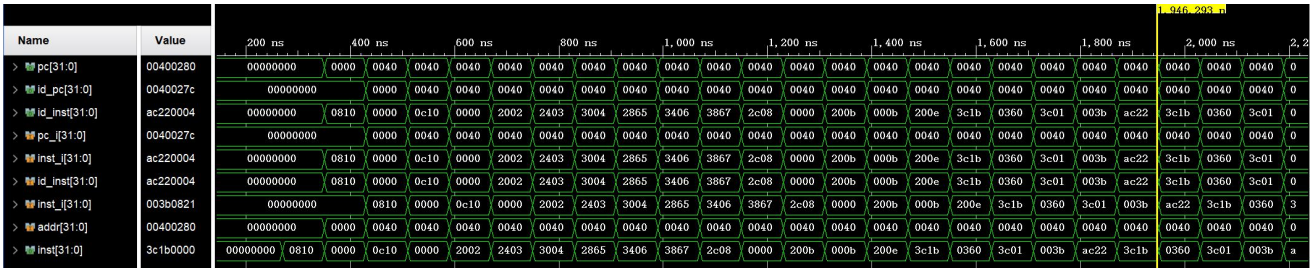
测试指令由 inst_rom.v 中指定路径的文本文件 test.txt 给出，存储于 inst_mem 当中。进行前仿真测试。

首先观察指令读取情况



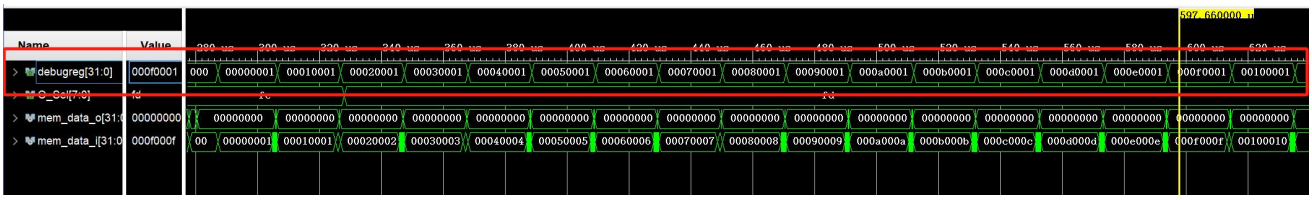
可以观察到指令可以正常读取，指令地址一次递增，取指正常

观察流经各个部件的指令和 pc 相关信号



可以观察到流水线能够正常流动

查看 debug_reg



可以观察到 debug_reg 前四位随着时钟中断计数不断增加，后四位保持不变，为 0x0001

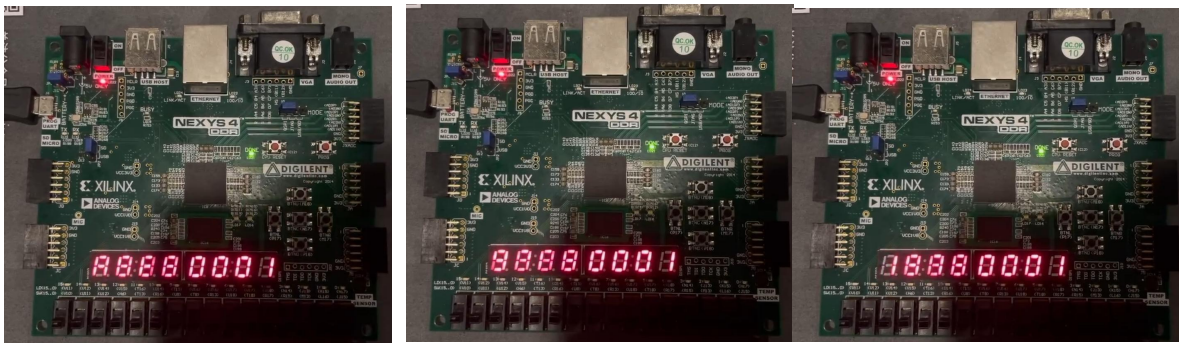
debug 寄存器的赋值部分为：

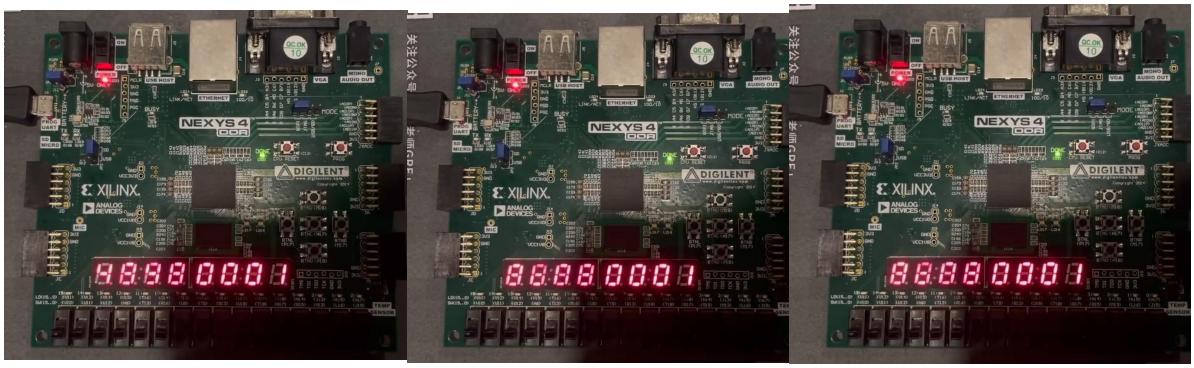
```
1.      assign debugreg={data_mem3[17'b0_0000_0000_0000_0000],
2.                       data_mem2[17'b0_0000_0000_0000_0000],
3.                       data_mem1[17'b0_0000_0000_0000_0000],
4.                       data_mem0[17'b0_0000_0000_0000_0000]};
```

波形图展示与 debug_reg 寄存器变化结果一致，数码管低 半字显示 0x0001，高半字随着时钟中断而计数，与预期一致。

5.3 下板验证

如图下板后，高半字随着时钟中断计数，低半字显示 0x0001，测试符合预期。





六、实验感想

通过该实验，我的对 CPU 工作流程的理解进一步加深。此次实验的主要任务是修改《自己动手写 CPU》雷思磊书中的 CPU 设计，使其能够在我们的 Nexys 4 开发板上顺利运行。在进行改造之前需要堆源代码有充分理解，需要尤其注意模块连接关系代码实现细节、CPU 工作流程逻辑等。在调整任何单一模块时，必须全面考虑其对整个系统的影响，注意是都需要修改控制信号等。

在本次实验中，我也丰富了我的 Debug 技巧，在 Vivado 的波形图界面选择 Scope -> Objects 的模块中的不同信号进行检测。同时在测试过程中我注意到了仿真时间的影响。在进行 debug_ram 信号观察时，由于仿真时间过短，导致我错误判断 CPU 连接存在问题，在信号连接上浪费了部分时间，最后通过调整 Vivado 模拟器的模拟时间解决问题。

本实验为后续操作系统移植实验的基础，希望在后续实验中，我能够堆本实验完成的支持 89 条指令的 CPU 进行进一步完善。

参考文献

[1] 秦国锋，王力生，陆有军，郭玉臣. 计算机系统结构实验指导，清华大学出版社,2019.

[2] 雷思磊. 自己动手写 CPU，电子工业出版社,2014.

[3] 张晨曦，王志英等. 计算机体系结构，高等教育出版社,2014.