实验报告成绩：	成绩评定日期：

2021～2022 学年秋季学期

A3705060050《计算机系统》必修课

课程实验报告

班级：人工智能 1902 班

组长：宿希琳

组员：孙博 孙嘉潞 王逍然

报告日期：2021.12.18

目录

一. 综述

1.成员工作量情况：

孙博：35% 孙嘉潞：25% 王逍然：15% 宿希琳：25%

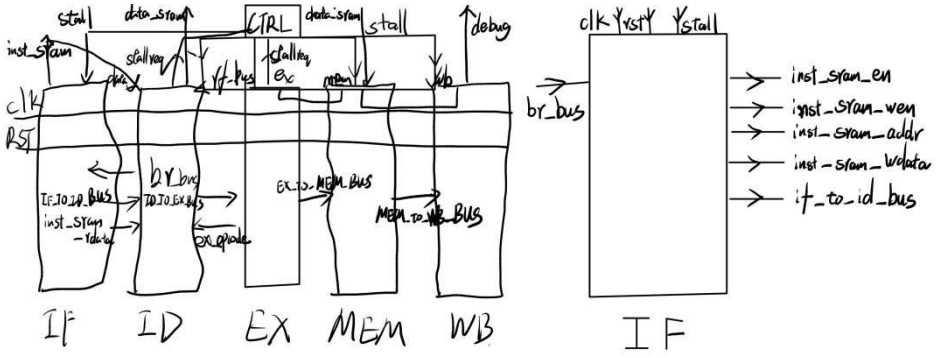
2.总体设计

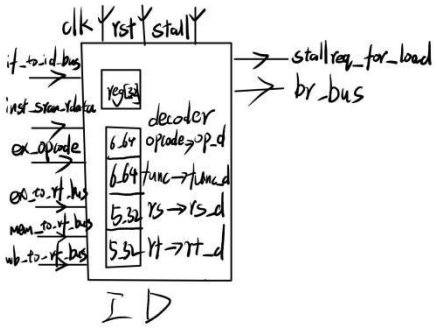
本次实验的基础目标是通过 64 个测试点。我们在完成这些测试之后，回顾完成这些指令添加与数据通路搭建的过程，可以把这个过程分为四个阶段：基础 ALU 运算操作以及基础数据通路的搭建阶段、跳转指令编写和跳转数据通路搭建阶段、乘除法器适配以及配套的 HILO 寄存器的搭建阶段、访存指令编写以及其所需的标识通路与数据通路的搭建部分。依次完成这四个部分即可通过 64 个测试点。

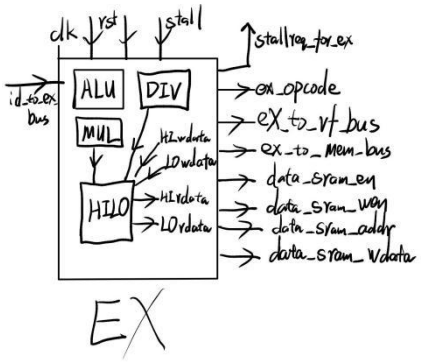
五段流水线分别是取指令周期（IF 段）、指令译码/读寄存器周期（ID 段）、执行/有效地址计算周期（EX 段）、存储器访问/分支完成周期（MEM 段）、写回周期（WB 段）。同时还有 CTRL.v 文件，用来控制周期的暂停等；ALU 模块，用来进行加减、移位、逻辑运算等等运算；乘除法器模块，用来完成乘除法操作，并且将结果存储与配套的 HILO 寄存器中（需要自行补全）；regfile 模块，拥有 32 个为一组的寄存器组，用来做高速的存取等等；还有一些译码器等等的小模块，组装成了整个流水线。

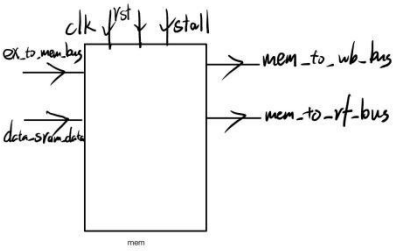
该流水线与内存进行需要读取或写入内存的数据相关交换以及指令信息的相关交换，并输出四条 debug 线用作提示信息。同时有 clk 时钟线和 rst 重置线进行流水线的启动及循环控制，这些是该流水线与外部的信息交换。在流水线内部，还有每一段传给下一段的 bus 总线、以及 WB 段传回 ID 段的寄存器写入总线（当然，从 EX 段的 ALU 计算出结果，到写回到寄存器，中间间隔了几个周期。为了解决这个数据相关问题，EX 段和 MEM 段同样有引回 ID 段的 rf_bus，以便在必要时迅速提供结果用于下面指令的执行，而不必等待数据传输到 WB 段再写回到 ID 段的 regfile 寄存器组）。还有几条 break 总线，用于控制跳转；还有更多隐藏于传输到下一段的总线中但是对流水线控制至关重要的总线。

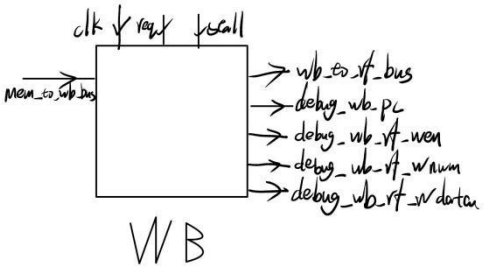
3.不同流水段之间的连线图











4.完成指令数量

算术运算指令 14 条:ADD,ADDU,ADDI,ADDIU,SUB,SUBU,SLT,

SLTU,SLTI,SLTIU,DIV,DIVU,MULT,MULTU.

逻辑运算指令 8 条:AND,ANDI,LUI,NOR,OR,ORI,XOR,XORI.

移位指令 6 条:SLLV,SLL,SRAV,SRA,SRLV,SRL.

分支跳转指令 12 条:BEQ,BNE,BGEZ,BLEZ,BLTZ,BGTZ,BGEZAL,BLTZAL,

J,JR,JAL,JALR.

数据移动指令 4 条:MFHI,MFLO,MTHI,MTLO.

访存指令 8 条:LB,LBU,LH,LHU,LW,SB,SH,SW.

共计指令数量 52 条。

5.程序运行环境及使用工具

运行环境为 vivado19.2

所使用的工具为个人笔记本电脑

二．单个流水段说明

IF 段

1.流水段的整体功能说明

IF 段，取指令周期。接收从 ID 段的跳转总线和从内存来的指令读取相关总线。如果不进行跳转，则输出下一条指令的 PC 值：当前 PC+4；如果选择跳转（由分支跳转指令控制），则输出下一条指令的 PC 值：跳转目标地址。输出到 ID 段。并且存储即将执行的下一条指令的 PC 值（下次未跳转的情况下）。主要作用就是读取即将执行的指令 PC 值，提供给 ID 段。

2.端口介绍

```
input wire clk,            //时钟线
input wire rst,            //重置线
input wire [`StallBus-1:0] stall,        //气泡线
input wire [`BR_WD-1:0] br_bus,     //跳转总线
output wire [`IF_TO_ID_WD-1:0] if_to_id_bus,     //IF_ID 总线
output wire inst_sram_en,                        //指令读信号线
output wire [3:0] inst_sram_wen,                 //指令写信号线
output wire [31:0] inst_sram_addr,               //指令地址线
output wire [31:0] inst_sram_wdata               //指令数据线
```

3.信号介绍

```
assign if_to_id_bus = {
    ce_reg,            //跳转标识位
    pc_reg             //下一个 PC 值
};


assign {
    br_e,          //跳转使能线
    br_addr        //跳转地址
} = br_bus;
```

4.包含的功能模块说明

该段内部未实例化其他结构。

5.结构示意图

ID 段

1.流水段的整体功能说明

ID 段，指令译码/读寄存器周期。接收从 IF 段传输来的 PC 地址。根据 ID 段是否插入气泡等等，找出即将执行的指令，并准备进行具体操作：根据指令的具体操作码来决定。添加指令的大部分操作也在 ID 段完成，包括但不限于：操作数的选取、寻址方式的设置、各种访存、跳转总线等线的装载配置等。在进行操作数选取时，我们需要检查是否有尚未存储入寄存器组但是马上需要被使用的数据，如果存在这样的数据需要从 EX 段、MEM 段、或 WB 段的 rf_bus 找出该数据并代替当前在寄存器存储的值，处理整合这三条 rf_bus 总线传输回的数据。同时需要根据指令类型，选取操作数寻址方式及扩展方式，传输给 EX 段，还有存储的目标寄存器，也要选好；如果是分支跳转指令，该段也需要完成分支条件的通过与否之判断，并根据结果装载跳转总线；如果是访存指令，该段也要完成对访存方式的初步分类，传递给下一段；以及诸如此类的大量操作，全部在 ID 段完成。在 ID 段实例化了拥有 32 个寄存器的寄存器组，用来存储指令执行所需的数据；以及四个解码器，用来解析指令的操作码、功能码、rs 寄存器地址、以及 rt 寄存器位置。该段需要传输给 EX 段大量数据：两个操作数、操作数的寻址与扩展方式、需要进行的运算操作、32 位指令本身以及关于访存控制需要的一些数据等等。这也是代码量最多的一段，读懂这段代码花费了我们很多精力。但是我们还是完成了。

2.端口介绍

```
input wire clk,                          //时钟线
input wire rst,                          //重置线
input wire [`StallBus-1:0] stall,   //气泡线
output wire stallreq,                                   //关于 load 需要的气泡
input wire [`IF_TO_ID_WD-1:0] if_to_id_bus,       //IF_ID 总线
input wire [31:0] inst_sram_rdata,                      //指令数据线
input wire [`WB_TO_RF_WD-1:0] wb_to_rf_bus,  //WB 寄存器写回线
output wire [`ID_TO_EX_WD-1:0] id_to_ex_bus,  //ID_EX 总线
output wire [`BR_WD-1:0] br_bus,                        //跳转总线
input wire [`EX_TO_RF_WD-1:0]ex_to_rf_bus,     //EX 数据旁路
input wire [`MEM_TO_RF_WD-1:0]mem_to_rf_bus,    //MEM 数据旁路
```

```
input wire[5:0] ex_opcode                          //关于 load 的 opcode
```

3.信号介绍

```
assign id_to_ex_bus = {
    data_ram_wen,        //访存写使能线
    inst_div,            //与 HILO 寄存器相关的八条指令
    inst_divu,
    inst_mult,
    inst_multu,
    inst_mfhi,
    inst_mflo,
    inst_mthi,
    inst_mtlo,
    id_pc,               // 158:127      //PC 值
    inst,                // 126:95       //指令内容
    alu_op,              // 94:83        //ALU 运算类型
    sel_alu_src1,        // 82:80        //ALU 操作数一取值方式
    sel_alu_src2,        // 79:76        //ALU 操作数二取值方式
    data_ram_en,         // 75           //访存读使能线
    data_ram_sel,        // 74:71        //访存字节读取方式
    rf_we,               // 70           //寄存器存入使能线
    rf_waddr,            // 69:65        //寄存器存入地址
    sel_rf_res,          // 64           //内存读取使能线
    rdata1,              // 63:32        //源操作数一
    rdata2               // 31:0         //源操作数二
};
```

4.包含的功能模块说明

该段内部实例化有四个解码器和一个 regfile32 位宽 32 个的寄存器组。

```
regfile u_regfile(
    .clk      (clk      ),      //时钟线
    .raddr1 (rs ),              //取操作数一：rs 寄存器地址
    .rdata1 (rdata1_xxl ),  //输出取出的数据
    .raddr2 (rt ),              //取操作数二：rt 寄存器地址
    .rdata2 (rdata2_xxl),   //输出取出的数据
    .we       (wb_rf_we       ),      //写使能线
    .waddr   (wb_rf_waddr   ),      //写地址
    .wdata   (wb_rf_wdata   )       //写数据
);

decoder_6_64 u0_decoder_6_64(
    .in   (opcode   ),              //译码：指令操作码
    .out (op_d )                    //译码结果
);

decoder_6_64 u1_decoder_6_64(
    .in   (func   ),                //译码：指令功能码
    .out (func_d )                  //译码结果
);

decoder_5_32 u0_decoder_5_32(
    .in   (rs   ),                  //译码：rs 寄存器地址
    .out (rs_d )                    //译码结果
);

decoder_5_32 u1_decoder_5_32(
```

```
    .in    (rt    ),                    //译码：rt 寄存器地址
    .out (rt_d )                        //译码结果
);
```

5.结构示意图

EX 段

1.流水段的整体功能说明

EX 段，执行/有效地址计算周期。该段接受了从 ID 段传来的总线数据、进行 ALU 和乘除法器计算后将处理结果传给 MEM 段，并且搭建了返回 ID 段的数据旁路，向内存输出数据请求信息，以及根据指令决定的部分气泡插入信息。该段进行了 ALU 模块和乘除法器模块的实例化。首先需要根据传来的标识位进行可能需要进行的扩展，然后将两个操作数传入 ALU 或者乘除法器进行计算。同时我在这个模块也完成了 HILO 寄存器模块的实例化以及其与乘除法器模块的对接。在 ALU 计算完成后，该段也进行了对从 ID 段传来的访存指令标识位的初步处理，并根据处理结果向内存请求数据。

2.端口介绍

```
input wire clk,                                 //时钟线
input wire rst,                                 //重置线
input wire [`StallBus-1:0] stall,               //气泡线
input wire [`ID_TO_EX_WD-1:0] id_to_ex_bus,         //ID_EX 总线
output wire [`EX_TO_MEM_WD-1:0] ex_to_mem_bus,      //EX_MEM 总线
output wire data_sram_en,                           //数据读总线
output wire [3:0] data_sram_wen,                    //数据写总线
output wire [31:0] data_sram_addr,                  //数据地址线
output wire [31:0] data_sram_wdata,                 //数据内容线
output wire [`EX_TO_RF_WD-1:0]ex_to_rf_bus,         //EX 数据旁路
output wire [5:0] ex_opcode,                        //关于 load 的 opcode
output wire stallreq_for_ex                         //关于除法器气泡
```

3.信号介绍

```
assign ex_to_mem_bus = {
    data_ram_sel_id,                    //访存指令使能线
    ex_pc,              // 75:44        //PC 值
    data_ram_en,        // 43           //内存读使能线
    data_ram_sel_ex,    // 42:39        //内存读取字节选择
    sel_rf_res,         // 38           //内存读取使能线
    rf_we,              // 37           //寄存器写使能线
    rf_waddr,           // 36:32        //寄存器写地址
    ex_result           // 31:0         //EX 段输出结果
};
```

4.包含的功能模块说明

该段实例化了 HILO 寄存器和 ALU 模块以及乘除法器。

```
regfile_HILO myregfile_HILO(
    .clk      (clk      ),          //时钟线
    .LO_we    (HI_we    ),          //LO 写使能线
    .HI_we    (LO_we    ),          //HI 写使能线
    .LOwdata(HI_rf_data),          //HI 写入数据
    .HIwdata(LO_rf_data),          //LO 写入数据
    .HIrdata(HIdata     ),         //HI 读出数据
  .LOrdata(LOdata       )          //LO 读出数据
);


alu u_alu(
    .alu_control (alu_op ),            //ALU 所要进行的运算
    .alu_src1      (alu_src1     ),    //ALU 操作数一
    .alu_src2      (alu_src2     ),    //ALU 操作数二
    .alu_result    (alu_result   )     //ALU 运算结果
```

```
);


mul u_mul(
    .clk            (clk                ), //时钟线
    .resetn         (~rst               ),//重置线
    .mul_signed (mul_signed         ),//有无符号标识位
    .ina            (alu_src1           ), //  乘法源操作数 1
    .inb            (alu_src2           ), //  乘法源操作数 2
    .result         (mul_result         ) //  乘法结果  64bit
);
div u_div(
    .rst                (rst                ),      //重置线
    .clk                (clk                ),      //时钟线
    .signed_div_i (signed_div_o ),      //有无符号标识位
    .opdata1_i      (div_opdata1_o      ),          //除法源操作数 1
    .opdata2_i      (div_opdata2_o      ),          //除法源操作数 2
    .start_i        (div_start_o        ),
    .annul_i        (1'b0           ),
    .result_o       (div_result         ), //  除法结果  64bit
    .ready_o        (div_ready_i        )
);
```

5.结构示意图

MEM 段

1.流水段的整体功能说明

MEM 段，存储器访问/分支完成周期。该段接受了来自 EX 段的总线以及从内存返回的数据，处理后将数据输出至 WB 段，并且搭建了返回 ID 段的数据旁路。该段进行的操作不多，

主要是对访存指令请求内存数据的最后处理。根据从 EX 段传来的标识位，以及从内存返回的数据，MEM 段选择了所需要的的字节数据，并写入结果，传输给 WB 段，然后存入寄存器。同时传输给 ID 段的数据旁路。

2.端口介绍

```
input wire clk,            //时钟线
input wire rst,            //重置线
input wire [`StallBus-1:0] stall,              //气泡线
input wire [`EX_TO_MEM_WD-1:0] ex_to_mem_bus,          //EX_MEM 总线
input wire [31:0]data_sram_rdata,                      //数据内容线
output wire [`MEM_TO_WB_WD-1:0] mem_to_wb_bus,   //MEM_WB 数据总线
output wire [`MEM_TO_RF_WD-1:0]mem_to_rf_bus       //MEM 数据旁路
```

3.信号介绍

```
assign mem_to_wb_bus = {
    mem_pc,       // 41:38          //PC 值
    rf_we,        // 37             //寄存器写使能线
    rf_waddr,     // 36:32          //寄存器写地址
    rf_wdata      // 31:0           //寄存器写数据     };
```

4.包含的功能模块说明

该段内部未实例化其他结构。

5.结构示意图

WB 段

1.流水段的整体功能说明

WB 段，写回周期。该段操作较少，接受了来自 MEM 段的数据，将其传输至 ID 段存入寄存器。同时根据传输来的其他标识性数据，传输出一些数据用于波形图生成，这对代码调试有较大帮助。该段操作不多，但为了流水线的模块化处理还是单独分成一段。

2.端口介绍

```
input wire clk,                  //时钟线
```

```
input wire rst,                         //重置线
input wire [`StallBus-1:0] stall,   //气泡线
input wire [`MEM_TO_WB_WD-1:0] mem_to_wb_bus,          //MEM_WB 总线
output wire [`WB_TO_RF_WD-1:0] wb_to_rf_bus,          //WB_寄存器总线
output wire [31:0] debug_wb_pc,                       //debug:PC 值
output wire [3:0] debug_wb_rf_wen,                    //debug:写使能线
output wire [4:0] debug_wb_rf_wnum,                   //debug:写寄存器地址
output wire [31:0] debug_wb_rf_wdata                  //debug:写入数据
```

3.信号介绍

```
assign wb_to_rf_bus = {
    rf_we,                  //寄存器写使能线
    rf_waddr,               //寄存器写地址
    rf_wdata                //寄存器写数据
};
```

4.包含的功能模块说明

该段内部未实例化其他结构。

5.结构示意图

三．组员的实验感受以及改进意见

孙博：

本次计算机系统实验为时两个月，是上大学以来第一次进行的底层设计。在这次的设计过程中，我学到的不仅是知识，我还认识到许多事情。这次设计使我的编程水平提高了一大步，使我充分的认识到合作的可贵。这次设计对我的综合能力是一次很好的锻炼，但是我必须承认自己的能力和知识还很肤浅。所以今后我的学习道路还是很漫长的。最后，在这里我要衷心的感谢我们的指导老师谢谢她的耐心指导和热心帮助。由于我水平有限，加之时间短暂，故学习计算机体系结构中还有许多不足之处，请老师批评指正，我会在以后的制作中不断改进，不断完善。我认为这两个月的计算机体系结构设计是给我们学习的一个大好机会，使我们在这样的机会里学到了一定的知识，毕竟理论要通过实践来锻炼，也只有自己参与了这样的一个锻炼，才能更好的发现自己的不足并加以改进和完善!通过这一次的计算机体系结构设计，使我能够提高分析问题、查阅资料、吸收新知识的能力，在分析解决问题时比以前有了很大的进步，一些常用的知识和一些常规的错误都能够解决。

两个月的课程设计结束了，在这次的课程设计中不仅检验了我所学习的知识，也培养了我如何去把握一件事情，如何去做一件事情，又如何完成一件事情。在设计过程中，与同学分工设计，和同学们相互探讨，相互学习，相互监督。学会了合作，学会了运筹帷幄，学会了宽容，学会了理解，也学会了做人与处世。这是我们专业课程知识综合应用的实践训练，这是我们迈向社会，从事职业工作前一个必不少的过程。通过这次设计，本人在多方面都有所提高，巩固与扩充了计算机系统等课程所学的内容，掌握设计的方法和步骤，提高了计算能力，绘图能力，熟悉了规范和标准，同时各科相关的课程都有了全面的复习，独立思考的能力也有了提高。在这次设计过程中，体现出自己单独设计的能力以及综合运用知识的能力，体会了学以致用、突出自己劳动成果的喜悦心情，从中发现自己平时学习的不足和薄弱环节，从而加以弥补。

通过本次课程设计的训练，让我对自己的专业有了更加感性和理性的认识，我们了解了底层逻辑元件设计的基本内容，掌握了计算机体系结构设计的主要程序和方法，增强了分析和解决工程实际问题的能力。同时，通过课程设计，还使我们树立正确的设计思想，培养实事求是、严肃认真、高度负责的工作作风，加强工程设计能力的训练和培养严谨求实的科学作风更尤为重要。

最后，我还要感谢于老师对我们的教导与帮助，感谢同学们的相互支持，与他们一起对一些问题的探讨和交流让我开拓了思路，也让我在课程设计时多了些轻松、愉快。

宿希琳：

通过此次课程设计，使我更加扎实的掌握了计算机体系结构以及流水线等方面的知识，在设计过程中虽然遇到了一些问题，但经过一次又一次的思考，一遍又一遍的检查终于找出了原因所在，也暴露出了前期我在这方面的知识欠缺和经验不足。实践出真知，通过亲自动手制作，使我们掌握的知识不再是纸上谈兵。过而能改，善莫大焉。在课程设计过程中，我们不断发现错误，不断改正，不断领悟，不断获取。最终的检测调试环节，本身就是在践行“过而能改，善莫大焉的知行观。

这次课程设计终于顺利完成了，在设计中遇到了很多问题，最后在老师的指导下，终于游逆而解。在今后社会的发展和学习实践过程中，一定要不懈努力，不能遇到问题就到要退缩，一定要不厌其烦的发现问题所在，然后一一进行解决，只有这样，才能成功的做成想做的事，才能在今后的道路上劈荆斩棘，而不是知难而退，那样永远不可能收获成功，收获喜悦，也永远不可能得到社会及他人对你的认可！计算机系统实验诚然是一门专业课，给我很多专业知识以及专业技能上的提升，同时又是一门讲道课，一门辩思课，给了我许多道，给了我很多思，给了我莫大的空间。同时，设计让我感触很深。使我对抽象的理论有了具体的认识。通过这次课程设计，我掌握了的识别和测试；熟悉了方法；以及如何提高的性能等等，掌握了的方法和技术，通过查询资料。

我认为，在这学期的实验中，不仅培养了独立思考、动手操作的能力，在各种其它能力上也都有了提高。更重要的是，在实验课上，我们学会了很多学习的方法。而这是日后最实用的，真的是受

益匪浅。要面对社会的挑战，只有不断的学习、实践，再学习、再实践。这对于我们的将来也有很大的帮助。以后，不管有多苦，我想我们都能变苦为乐，找寻有趣的事情，发现其中珍贵的事情。就像中国提倡的艰苦奋斗一样，我们都可以在实验结束之后变的更加成熟，会面对需要面对的事情。回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，可以说得是苦多于甜，但是可以学到很多很多的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。

通过这次课程设计使我懂得了理论与实际相结合是很重要的，只有理论知识是远远不够的，只有把所学的理论知识与实践相结合起来，从理论中得，才能真正为社会服务，从而提高自己的实际动手能力和独立思考的能力。在设计的过程中遇到问题，可以说得是困难重重，但可喜的是最终都得到了解决。实验过程中，也对团队精神的进行了考察，让我们在合作起来更加默契，在成功后一起体会喜悦的心情。果然是团结就是力量，只有互相之间默契融洽的配合才能换来最终完美的结果。

孙嘉潞：

本次实验课让我体验到了 cpu 的设计过程。对于本次实验：自己动手写 cpu 来说，我认为自己和组员们通过不懈的努力，已经完成了任务。参与这次课程设计的目的，对我来说是进一步了解计算机组成与体系结构，因为本专业考研大多会遇到一门叫计算机组成原理的课，感觉对它完全没有概念，于是想着干脆通过自己写一个 cpu 来增强一下对计算机的理解。学长推荐的书是《自己动手写 cpu》，看了一下感觉非常适合我，便开始着手仔细看了。通过这次实验我的首要感想是：原来 cpu 并没有那么神秘！在这里我想讲一下两方面的收获：cpu 的结构与设计以及计算机组成与体系结构。

我之前对 cpu 有很多好奇和疑惑：以前的我只知道 cpu 是按照时钟的频率的工作的，但是不知道为什么要由时钟来控制，以及如何控制等进行了这次实验之后，我知道了这些问题的答案。首先，有时钟，是为了保证 cpu 内部的正常秩序，“到了什么时间就做什么事”，没有一个良好的秩序，一个团队的效率是不可能高的，甚至乱作一团，导致不能工作。那么，时钟是如何控制（组织）cpu 内部的工作的呢？时钟会产生一个周期信号，我们可以规定在信号的每一个上升沿 or 下降沿开始执行一些操作，cpu 中的不同模块能同时进行操作，但 cpu 中的一个模块在同一个时刻只能做一件事，因此就需要这样来保证各个模块之间不冲突，各个功能正常执行。那么是不是一直提高时钟的频率，就能一直无限制地提高 cpu 的执行速度呢？显然是不可能的。第一个事实是由于 cpu 电路中的信号是有延迟的，一个操作的结果不可能立刻反映在输出电路上，如果无限制提高时钟频率，那么会导致“运算”跟不上时钟，从而导致 cpu 内部的秩序又消失了，变得混乱。第二就是由于电路中电子的运动是要消耗能量的，执行地越快，单位时间内消耗的能量就越多，发热量就越大，散热就会跟不上，从而导致内部电路“失效”，电脑蓝屏、死机。更重要的是，时钟频率并不是唯一决定处理器性能的东西。

cpu 的结构及设计：cpu 的本质就是一个只会“解释指令”，并且死板地执行指令的没有感情的机器，有一个比喻说得好：硬件是计算机的躯体，OS 才是灵魂。它的主要结构（或者说最基本的结构）包括译码器、ALU 以及各种寄存器。我们会给具体指令集中的指令规定固定的格式，以便让译码器理解代码，其实原理很简单，cpu 译码的过程相当于就是查字典，在规定的有限的指令集内找到与输入的机器码相匹配的情况，并执行相应的操作。ALU 的中文名是算术逻辑单元，顾名思义，主要是执行算术/逻辑操作的单元，计算机的各种“计算”操作都是在这里完成的。寄存器是用来保存运算的输入数据、中间结果 and 结果的存储单元，其位于 cpu 内部，访问速度最快（在计算机体系中）。但也有一些特殊的寄存器并不用于计算，比如 PC 程序计数器，它的值是下一个将要取到的指令的地址。cpu 中的各个模块按照时钟来运行的， 为了进一步提高效率，人们使用了流水线结构，类似于工厂众多的流水线，将一条指令（商品）的执行过程分为多个步骤（或阶段） ，每一个步骤由一个专门的模块来负责， 这个模块的任务就是接收从上一个阶段传来数据，经过它自己的处理后，再传递给下一个模块。本次实验要实现 的 cpu 采用的是五级流水线，将一条指令分为五个步骤：取指、译码、执行、访存（RAM）、回写（寄存器）。 具体实现流水线的方式就是通过组合电路和时序电路的组合级联而，由组合电路完成具体的操作，由时序电路完成按照时钟传递数据的任务，以保证指令的有序执行。同时，由于采用了流水线结构，也会相应地带来相邻指令间的相关问题，简单来说就是：假如一条指令的结果是在最后一个流水线阶段才写入的，此时流水线的前几个阶段已经在处理后面几条指令了，如果这几条指令需要的结果正好是该指令将要写入的（但是还没写入），那么就会产生数据相关问题。这也是本次实验中要首先解决的问题。

什么是计算机体系结构？有的教材中将其定义为：计算机体系结构是程序员所看到的计算机的属性，即概念性结构与功能特性对于不同层次的程序员来说，他们看到的计算机的属性也不相同。例如，对现在的许多程序员来说，他们看到的计算机的属性大多数是操作系统（OS）提供的，而对于 OS 程序员来说，他们看到的体系结构就是 ISA（提供的。至于计算机组成，就是由五个经典的部分组成：输入、输出、存储器、数据通路（运算器）、控制器。任何一个 ISA 都会提供给底层程序员几个类别的指令，《自己动手写 cpu》一书实现的 Open MIPS 就包括了：逻辑指令、位移指令、（数据）移动指令（寄存器之间相互移动）、算术指令、转移指令、加载存储指令（寄存器与 ram 和 rom 之间移动）、协处理器访问指令、异常相关指令。ISA 所提供的指令类别需要保证通过不同指令的组合就能完成任何程序的任何逻辑，现在同一个类（RISC 或 CISC）中的不同 ISA 的指令类别应该大同小异了。具体的指令设计可是一个大学问，这需要考虑到指令有哪些格式，什么样功能的指令是必要的，什么是冗余的，如何设计执行过程的方法才能提升指令的执行效率。不同的指令执行所需要的时间也不一定一样，在 Open MIPS 中，大部分指令执行阶段都只需要一个时钟周期，少部分需要两到三个，最花时间的是除法指令。实现同一个功能可以由多种不同的指令组合完成，如何提高程序的执行效率，也就可以通过寻找所需最短时间的指令组合来实现，这个过程是由编译器实现的。同时这也给我们编写高级语言程序的时候一些启示：了解一些高级语言语句的汇编实现是有用的，因为这可以让我们判断哪一种实现方法更有效率（所需的时钟周期更少）/汇编语句更少（节省内存空间），这

是让我们编写出更好的程序的一种技巧（个人观点）。举个例子，调用函数操作的汇编语言实现一般都是跳转指令配合标签，同时会用到栈（将调用前的寄存器值压入栈），如果使用递归的话，就会多次调用函数，多次将寄存器的值压入栈，如果递归深度太深，那么栈就可能溢出，导致程序错误，同时相比非递归函数，这个压入栈的操作会被执行很多次，就浪费了时间，同时也浪费了空间。这也是为什么一般不轻易采用递归结构的原因。

最后，总的来说，这次实验实现的 cpu 让我们受益良多，内容也非常有用，同时，看这本书的收获也挺多，可以算是对 cpu 和计算机硬件有了初步的系统了解。

王逍然：

在整个实验过程中，为了通过 passpoint1，我们首先需要结局数据通路问题，也就是完成数据相关。平心而论，这不是一个十分简单的事情，对于我本人而言，之前完全没有底层的相关知识，实验所需要的无论是 verilog 语言还是 vivado 环境都和我在之前的学习生活中所学习的语言有着相当大的差距。尽管在此之前经历了一些基础的类似模 60 计数器的训练，能够熟练地掌握一门语言也是十分困难的。同时，由于实验的要求，我们实际上是在助教的设设计思路的基础上不断完善，这在一定程度上减轻了我们的前期负担，但同时，去理解助教的代码的含义婺花费了我们一些时间。在这个过程中，我充分意识到越是复杂的问题越需要整个小组所有成员在一起讨论分析。有很多问题，即使你查阅资料，在图书馆苦思冥想一下午都无法解决，却可能在和队友的交流与思想的碰撞中仅仅花费一个小时就有了思路。也是由于这个原因，在队友们的协助和帮助下，我们成功的解决了初步的数据相关问题。在这之后就需要进行指令的添加。由于实验所使用的是龙芯大赛的环境，添加指令这项工作在我个人看来，尽管比解决数据相关要简单，但是，由于对指令的不熟悉，在添加初期查找指令过程中还是花费了很多努力。在大家的不懈努力下，我们终于成功添加了所需要的运算指令和位移指令。成功通过 passpoint1 的时候，大家那发自内心的喜悦让我真切的体会到团队协作解决难题的成就感。

在之后，我们不断地添加至零，同时，面对 load 读取数据问题时，由于我对于插入空铺相关知识掌握的不够熟练，进展一度陷入停滞。但是，在队友的帮助下，我们还是顺利的解决了文图，来到了下一个难关：passpoint36。在这个测试点，我们需要添加数据转指令，同时需要添加两条新的总线并且编写跳转方式。这和之前进的的工作有一定程度的不同，对于流水线技术理解的不够透彻也让我在添加总线的时候犯了一些错误，导致了许多时间用在了无用的调试和改错中。在这个过程中，我对于波形图有了更加深刻的印象。在此之前，我很多的程序编写和设计还是印象流，并未明确的体会到波形图对于本次实验的重要性。在面对这个问题的时候，波形图起到了十分重要的作用，与此同时，学习和理解波形图也是一个困难的过程。值得庆幸的是，在队友的帮助下，我们终于解决了这个问题。这个测试点通过后，我们面对 passpoint44 时需要引入新的寄存器 HI 和 LO 为乘除法其做准备，也又一次遇到了数据相关和添加指令问题。但是，在长时间的工作下，我们对于 verilog 语言，乃至整

个实验的理解有了之前难以想象的进步，对于实验中遇到的问题处理的也越来越得心应手。最终，在我们小组的不懈努力下，成功的通过了 64 个测试点，完成了实验的基本需求。

对我个人而言，其实是有一些羞愧的情绪。在本次实验中，由于个人能力的不足，我更多地是承担一些边角料的工作，未能给队伍提供关键性的帮助。但我认识到自己不足的同时，也更加深刻的认识到分工合作，互相帮助，讨论交流对于解决现实问题的重要。或许在未来，面对着越来越复杂的问题，相比于个人英雄主义式的灵光乍现，团队成员间的精诚合作才是解决问题的最好办法。