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

组长：李嘉乐

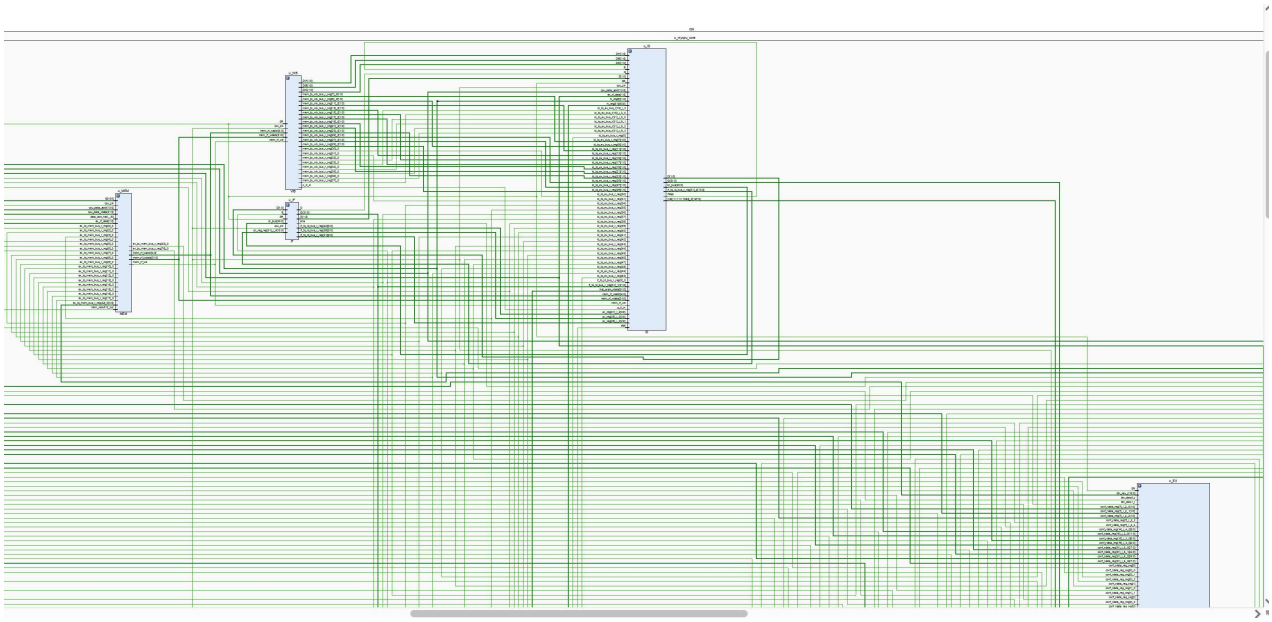
组员：周林锋，刘腾浩

报告日期：2021.12.18

工作量分配：

李嘉乐：50%，周林锋：30%，刘腾浩：20%。

总体设计：



本次实验添加的指令：

处理器需要实现的指令包括除 4 条非对齐指令外的大部分 MIPS I 指令以及 MIPS32 中的 ERET 指令，有 14 条算
术运算指令、8 条逻辑运算指令，6 条移位指令、8 条分支跳转指令、4 条数据移动指令 12 条访存指令共计 52 条。下面分类给出各部分指令的简要功能介绍。

表 3-1 算术运算指令

ADD rd, rs, rt 加（可产生溢出例外）
ADDI rt, rs, immediate 加立即数（可产生溢出例外）
ADDU rd, rs, rt 加（不产生溢出例外）
ADDIU rt, rs, immeidate 加立即数（不产生溢出例外）
SUB rd. rs, rt 减（可产生溢出例外）
SUBU rd, rs, rt 减（不产生溢出例外）

SLT rd, rs, rt 有符号小于置 1
SLTI rt, rs, immediate 有符号小于立即数设置 1
SLTU rd, rs, rt 无符号小于设置 1
SLTIU rt, rs, immediate 无符号小于立即数
设置 1
DIV rs, rt 有符号字除
DIVU rs,rt 无符号字除
MULT rs, rt 有符号字乘
MULTU rs, rt 无符号字乘

表 3-2 逻辑运算指令 指令名称格式 指令功能简述

AND rd, rs, rt 位与
ANDI rt, rs, immediate 立即数位与
LUI rt,immediate 寄存器高半部分置立即数
NOR rd, rs, rt 位或非
OR rd, rs, rt 位或
ORI rt, rs, immediate 立即数位或
[1] 请注意虽然是无符号比较，但是立即数仍是进行有符号扩展。4
XOR rd, rs, rt 位异或
XORI rt, rs, immediate 立即数位异或

表 3-3 移位指令 能简述

SLL rd, rt, sa 立即数逻辑左移
SLLV rd, rt, rs 变量逻辑左移
SRA rd, rt, sa 立即数算术右移
SRAV rd, rt, rs 变量算术右移
SRL rd, rt, sa 立即数逻辑右移
SRLV rd, rt, rs 变量逻辑右移

表 3-4 分支跳转指令 简述

BEQ rs, rt, offset 相等转移
BNE rs, rt, offset 不等转移
BGEZ rs, offset 大于等于 0 转移
BGTZ rs, offset 大于 0 转移
BLEZ rs, offset 小于等于 0 转移
BLTZ rs, offset 小于 0 转移
BLTZAL rs, offset 小于 0 调用子程序并保存返回地址
BGEZAL rs, offset 大于等于 0 调用子程序并保存返回地址
J target 无条件直接跳转
JAL target 无条件直接跳转至子程序并保存返回地址
JR rs 无条件寄存器跳转
JALR rd, rs 无条件寄存器跳转至子程序并保存返回地址下

表 3-5 数据移动指令 5 指令功能简述

MFHI rd HI 寄存器至通用寄存器
MFLO rd LO 寄存器至通用寄存器
MTHI rs 通用寄存器至 HI 寄存器
MTLO rs 通用寄存器至 LO 寄存器

表 3-7 访存指令 指令功能简述

LB rt, offset(base) 取字节有符号扩展

LBU rt, offset(base) 取字节无符号扩展

LH rt, offset(base) 取半字有符号扩展

LHU rt, offset(base) 取半字无符号扩展

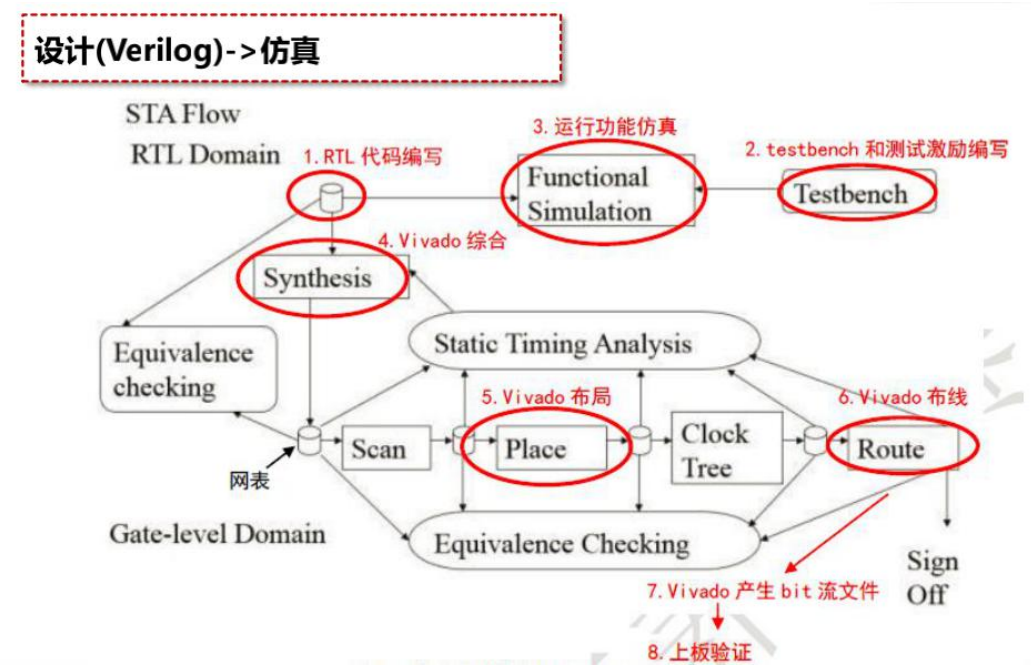
LW rt, offset(base) 取字

SB rt, offset(base) 存字节

SH rt, offset(base) 存半字

SW rt, offset(base) 存字

程序运行环境及使用工具：



单个流水线的说明：

IF:

整体功能说明：

计算 pc 值和下一条指令的 PC 值，并发给 id 段。

端口和信息介绍：

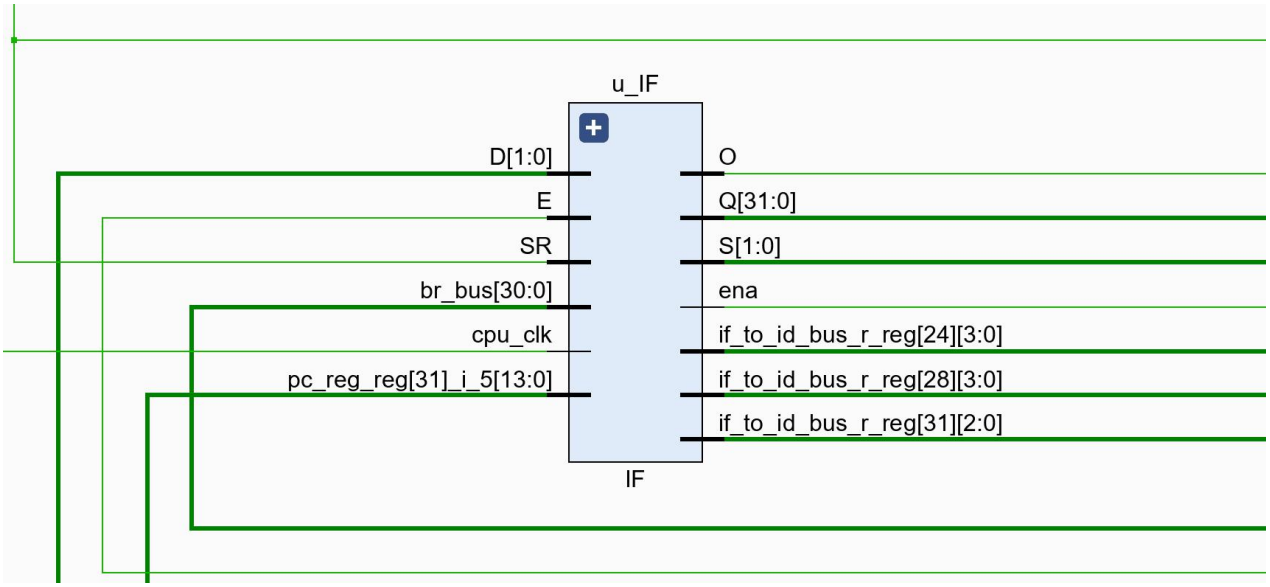
input wire clk,

input wire rst,
input wire [`StallBus-1:0] stall,
input wire [`BR_WD-1:0] br_bus,
output wire [`IF_TO_ID_WD-1:0] if_to_id_bus,
output wire inst_sram_en,
output wire [3:0] inst_sram_wen,
output wire [31:0] inst_sram_addr,
output wire [31:0] inst_sram_wdata
说明:
input wire clk,传递时钟信息。
input wire rst,复位信号。
input wire [`StallBus-1:0] stall,插入气泡使用

包含的功能模块说明:

无

结构示意图:



ID:

整体功能说明:

指令译码。判断是否使用数据前推，将 WB 段传回的数据写入寄存器。选择操作数来源。判断分支指令，并计算跳转目的地址。写入 hi lo 寄存器的值。判断 load 造成的数据相关是否需要暂停。

端口和信号介绍：

1.添加乘除法器时，需要定义乘除法结果寄存器 Hi 和 Lo，本组这两个寄存器添加位置在原框架所定义的 regfile 模组内部。

```
always @ (posedge clk) begin
        if (we && waddr!=5'b0) begin
            reg_array[waddr] <= wdata;
        end
        if (hi_we) begin
            hi <= hi_i;
        end
        if (lo_we) begin
            lo <= lo_i;
        end
    end
```

regfile 中定义的 Li 和 Lo 寄存器与 reg_array 类似，在时钟信号上升时，若使能线为 1 则完成赋值。

2.定义 66 位宽的线路用来将 EX 段的乘除法结果送回 ID 段并写入 Hi、Lo 寄存器，其中前两位是 Hi、Lo 使能线，其值由 op_mul | op_div 和 op_mul | op_div 两式决定，即只要进行乘除法操作就需要激活 Hi、Lo 使能线，后 64 位为乘除法结果，若为乘法则 63：32 位为乘法结果的高位部分应存入 Hi 寄存器，31：0 位为乘法结果的低位部分应存入 Lo 寄存器；若操作为除法则 63：32 位为商应存入 Hi 寄存器，31：0 位为余数应存入 Lo 寄存器。

```
assign {
            hi_we_exr,
            lo_we_exr,
            hi_i_exr,
            lo_i_exr
    } = mul_div_res_bus;
    assign hi_we = inst_mthi ? 1'b1 : hi_we_exr;
    assign lo_we = inst_mtlo ? 1'b1 : lo_we_exr;
    assign hi_i = inst_mthi ? rdata1 : hi_i_exr;
    assign lo_i = inst_mtlo ? rdata1 : lo_i_exr;
```

ID 段得到的上述乘除法结果并不能直接使用，因为指令集中还有 mthi、mtlo 两条指令也需要写 Hi、Lo 寄存器。因此需要将 mthi、mtlo 的指令译码结果用来作为选择器的控制线，若是这两条指令则也需要将 Hi 或 Lo 使能线置为 1，同时向 Hi 或 Lo 寄存器写入 rs 的值。

```
assign rdata1 = ((rs==ex_rf_waddr)&&(ex_rf_we)) ? ex_rf_data :
                        ((inst_mfhi & hi_we) ? hi_i :
                        ((inst_mfhi & ~hi_we) ? hi_o :
                            (((rs==mem_rf_waddr)&&(mem_rf_we)) ? mem_rf_wdata :
                            (((rs==wb_rf_waddr)&&(wb_rf_we)) ? wb_rf_wdata: rdata1_r))));
assign rdata2 = ((rt==ex_rf_waddr)&&(ex_rf_we)) ? ex_rf_data :
                        ((inst_mflo & lo_we) ?    lo_i :
                          ((inst_mflo & ~lo_we) ? lo_o :
                              (((rt==mem_rf_waddr)&&(mem_rf_we))? mem_rf_wdata :
                               (((rt==wb_rf_waddr)&&(wb_rf_we)) ? wb_rf_wdata : rdata2_r))));
```

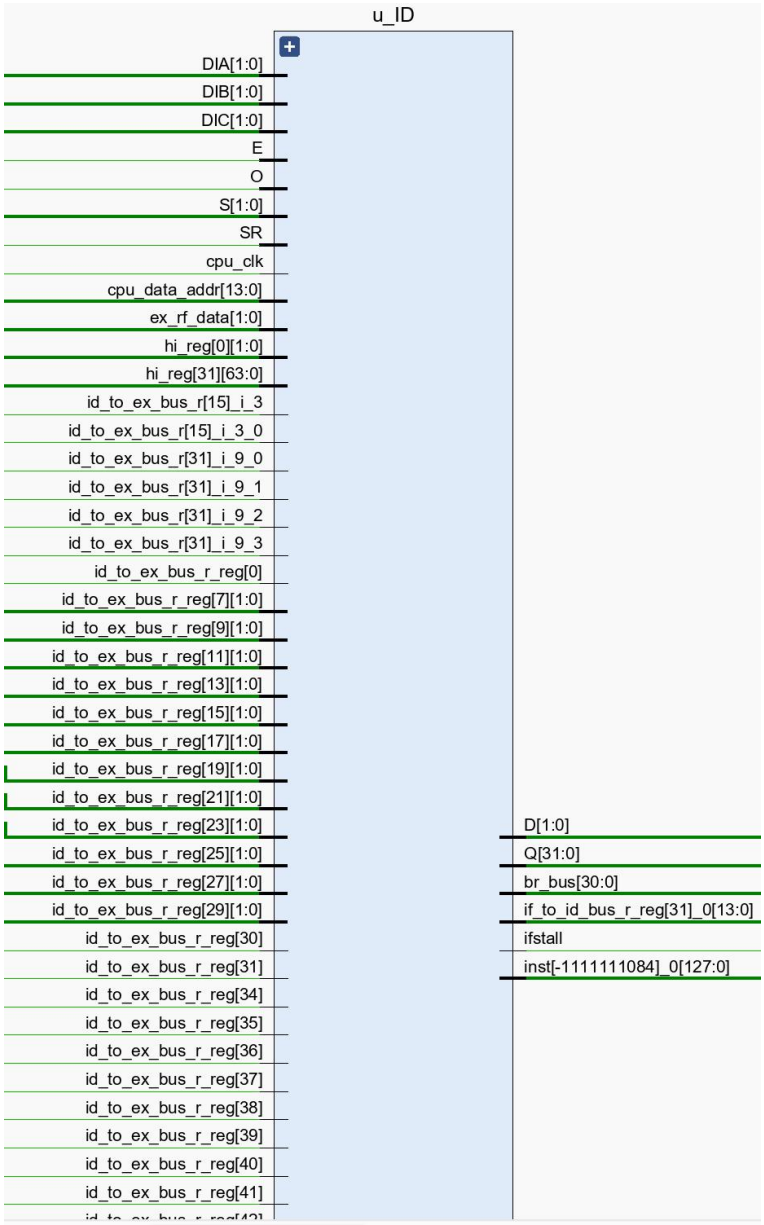
由于 Hi、Lo 寄存器也可能存在数据相关，因此需要将其加入数据相关判断逻辑中，且应置为高优先级，否则在第 53 点处会出现 WB 段回传数据写入地址与 mflo 指令的写入相同并将 Lo 值替换为 wb_result 的情况导致出错。

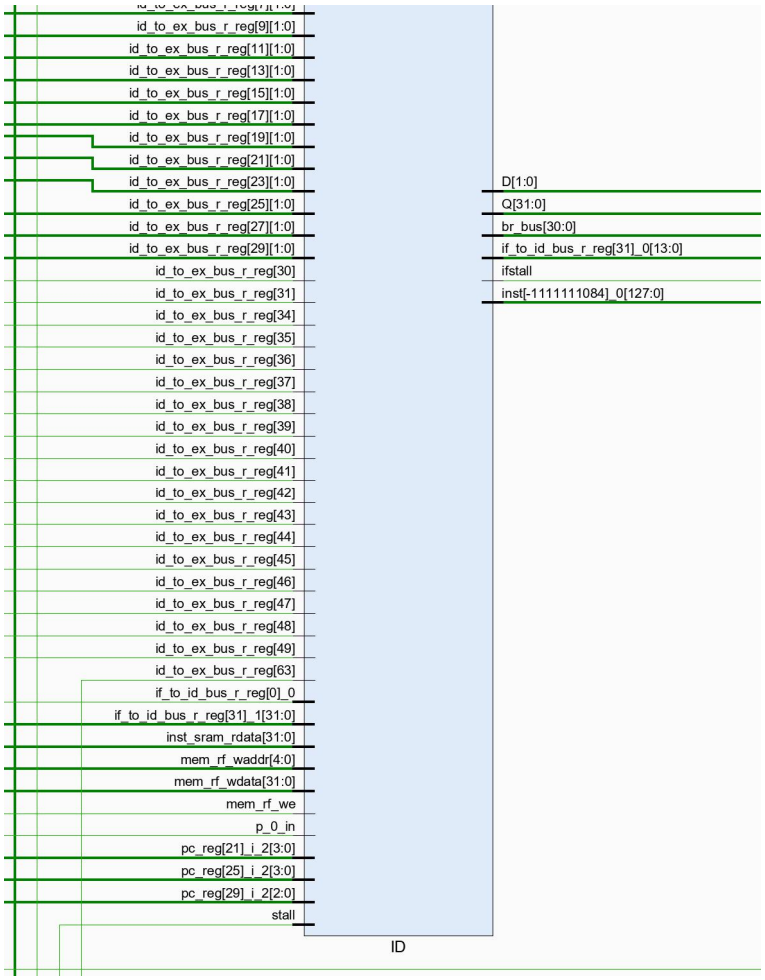
包含的功能模块说明：

Regfile,包含 32 个寄存器和 hi lo 寄存器。64 位译码器。32 位译码器

。

结构示意图：





X:

EX：

整体功能说明：

根据 ID 选择的操作数进行运算，判断是否因乘除法造成的暂停。计算四位访存使能线的值，向内存写入数据。

端口和信号介绍：

1.除法器运行时需要暂停 32 周期，故需要修改 ctrl 模块，添加处理由 ex 段发出的暂停申请的逻辑电路

```
else if(stallreq_for_ex) begin
                stall = 6'b00_1111;
        end
```

除法器未完成运算前需要 ID 段 PC 值保持不变, 以保证 ID 和 EX 段均为当前执行的除法指令, 同时, 与 Load 导致的数据相关暂停不同的是，ID 段虽然 PC 值不变，但不能暂停工作，因为 EX 段需要由 ID

段传来的指令以继续执行除法操作，故只需要暂停 IF 段，stall 信号置为 001111。

由于 alu 中没有乘除法操作，故需要修改 id_to_ex_bus 的线宽，增加三位用来标识乘除发指令已经乘除法操作是否有符号。

```
assign mul_div_res_bus = {
        (op_mul | op_div),
        (op_mul | op_div),
        (op_mul ? mul_result : div_result)
    };
```

2.指令 mfhi、mflo 需要将 hi 值或 lo 值写入 rt 寄存器，因此在 EX 段添加 ex_result 多路选择器，如果指令为 mfhi 则 ex_result 为 hi 值, 如果指令为 mflo 则 ex_result 为 lo 值, 否则 ex_result 为 alu_result。

```
assign ex_result = mfhi ? rf_rdata1 : (mflo ? rf_rdata2 : alu_result);
```

3.

添加 inst_lb, inst_lbu, inst_lh, inst_lhu, inst_sb, inst_sh 指令需要修改访存使能线控制逻辑，由于通常情况下的访存是以 32 位字为单位，故访存接口的寻址方式也是以 32 位字为单位，逻辑上可以理解为读写头是连续四个字节的（实际上并没有读写头结构）且只能将最低位对准 4 的整数倍的地址，若所寻地址不是 4 的整数倍，则读写头会自动找到包含该地址的 32 位字并将最低位对准该字的最低位。这种情况下，所寻地址对 4 取模的值即为要激活的读写头中的字节编号，在实际操作中，可以取所寻地址的最低两位作为对 4 取模的值，例如所寻地址的最后两位为 01，说明需要将四位使能线编号为 1 的使能线置 1，以下代码即为上述逻辑的实现。

```
assign data_ram_b = (ex_result[1:0] == 2'b00) ? 4'b0001 :
                                (ex_result[1:0] == 2'b01) ? 4'b0010 :
                                 (ex_result[1:0] == 2'b10) ? 4'b0100 : 4'b1000;
```

其中 data_ram_b 表示若只存入一个字节所需的四位使能线激活方式，如果要存入的数据为半字，则地址最低两位只有 00 和 10 两种情况，00 对应需要存入读写头对准位置的低半字，10 对应需要存入读写头对准位置的高半字，以下是代码实现。

```
assign data_ram_h = (ex_result[1:0] == 2'b00) ? 4'b0011 : 4'b1100;
```

如果是 sw 指令则需要激活全部四根使能线，使能线计算方式的选择需要用到以下代码所定义的多路选择器。

```
assign data_ram_selected = {4{data_ram_sel[0]}} & data_ram_w
                                    | {4{data_ram_sel[1]}} & data_ram_h
                                     | {4{data_ram_sel[2]}} & data_ram_b;
assign data_sram_wen = real_data_ram_wen ? data_ram_selected : 4'b0;
```

data_ram_sel 使用的是 id_to_ex_bus 的 74：71 位，data_ram_sel[0]对应字寻址，data_ram_sel[1]对应半字寻址 data_ram_sel[2]对应字节寻址，data_ram_sel[3]对应作为 load 指令时取半字或字节后是否选择有符号扩展，data_ram_selected 即为选择出的四位使能线的值，而后需要位于 id_to_ex_bus 最高位的 real_data_ram_wen 选择信号控制是否激活使能线，该值直接对应于 store 系列指令线，由于需要增加上述选择信号，必须再次修改 id_to_ex_bus 的线宽。

data_sram_wdata 的值为 cpu 要向 mmu 的写入寄存器传递的数据, 如果写入的数据为字节或半字, 需要将该字节重复四次或将半字重复两次以填满 mmu 的写入寄存器（实验时观察似乎 mmu 会默认取 data_sram_wdata 的最高 8 位或 16 位，但保险起见仍然设置了足够的冗余度），防止 mmu 取到错误数据，如下所示。

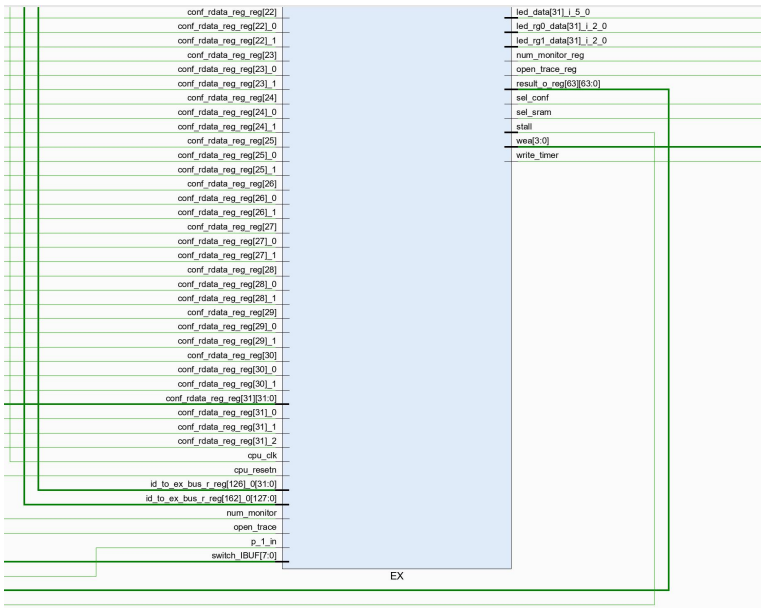
```
assign data_sram_wdata = ({32{data_ram_sel[0]}} & rf_rdata2)
                                    | ({32{data_ram_sel[1]}} & {2{rf_rdata2[15:0]}})
```

| ({32{data_ram_sel[2]}} & {4{rf_rdata2[7:0]}});

包含单元模块：

ALU，乘除法单元（mul_div）

结构示意图：



MEM：

整体功能说明：

读取内存访存结果，并根据 EX 提供的访存使能线的信息（复用了读和写的使能信息）计算 load 类指令结果。

端口和信号介绍：

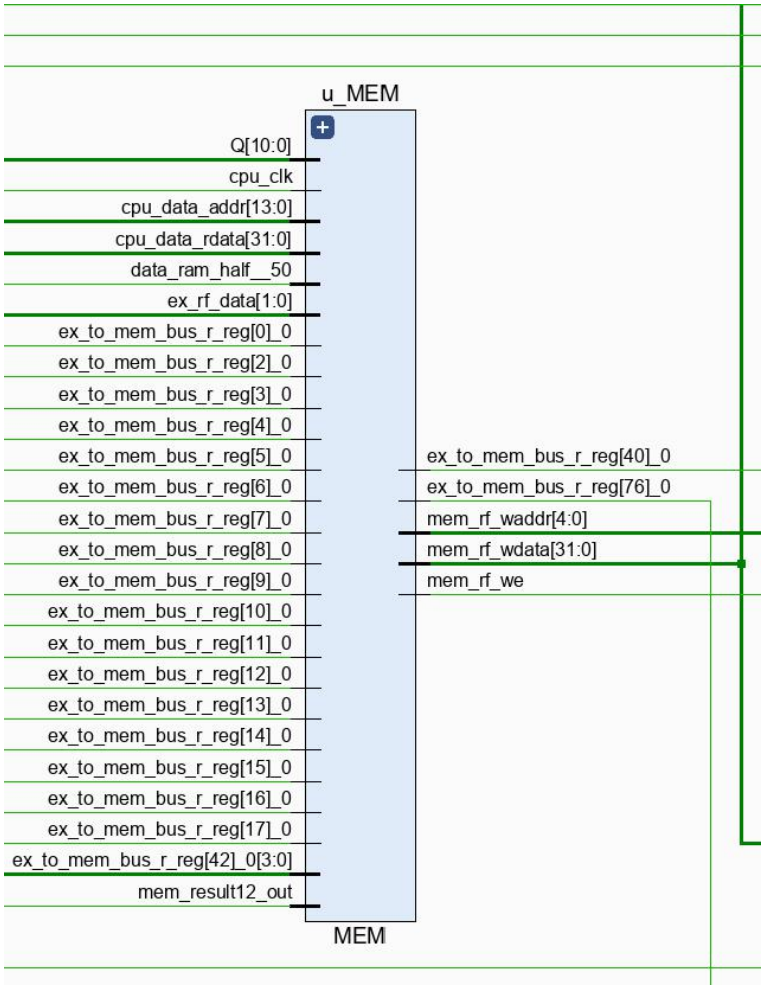
load 系列指令的字节选择与 store 系列类似，但读写头并不能直接按照使能线选择读取那些字节，只能将对准的四个字节数据全部储存于 data_sram_rdata 寄存器中，并在 cpu 中完成选择。由于使能线计算方式与 store 系列指令完全相同，没有必要在 MEM 段中重复计算，只需要修改 ex_to_mem_bus 的线宽，并将 EX 段计算并选择好的 data_ram_selected 值置于 ex_to_mem_bus 的 42：39 位传送至 MEM 段即可，同时需要将 data_ram_sel 置于最高四位，具体控制逻辑由下列代码展示。

assign data_ram_half = (data_ram_selected == 4'b1100) ? data_sram_rdata[31:16] : data_sram_rdata[15:0];

assign data_ram_byte = (data_ram_selected == 4'b1000) ? data_sram_rdata[31:24] :

```
                                    (data_ram_selected==4'b0100)?
                                    data_sram_rdata[23:16] :
                                    (data_ram_selected==4'b0010)? data_sram_rdata[15:8] :
                                            data_sram_rdata[7:0];
    assign mem_result = ({32{data_ram_sel[0]}} & data_sram_rdata) |
      ({32{data_ram_sel[1]}}&{{16{data_ram_sel[3]}}& {16{data_ram_half[15]}},data_ram_half})|
            ({32{data_ram_sel[2]}}&{{24{data_ram_sel[3]}}&
  {24{data_ram_byte[7]}} ,data_ram_byte})
```

结构示意图：



WB：

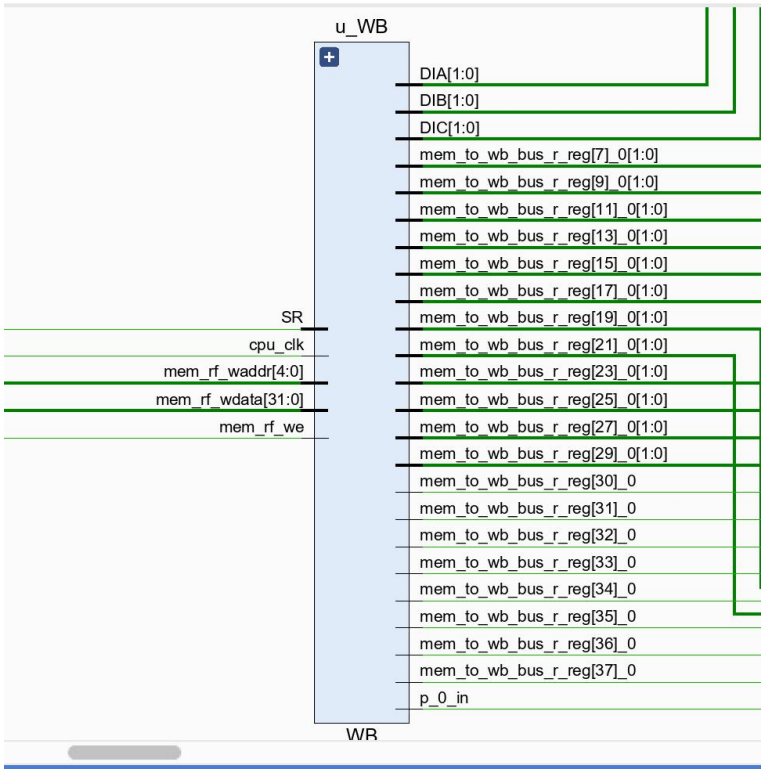
整体功能说明：

把从 Mem 传来的结果传到 id 段。

包含功能模块：

无。

结构示意图：



组员的实验感受：

李嘉乐实验心得体会：

通过计算机系统本次实验，我不仅较为扎实的掌握了 CPU 结构及设计方面的知识，而且还对如何解决一个困难的问题，以及该以何种姿态面对困难问题有了更加深刻的领会，这些经验让我收益良多。

这个问题是一个涉及面非常广泛的问题，开始面对这个问题的时候，我们小组的每个成员都还对问题的真实性没有任何的感知，不知道该干什么，以及从哪里开始做事情。我们在开始的时候经历了不少的挫败。开始总是困难的，但是开始了之后我们就可以慢慢的把事情做的越来越好，我们在做实验的过程中，慢慢的熟悉了开发环境 vivado 的使用，慢慢的熟悉了 Verilog 语言，慢慢熟悉了 cpu 的结构。在这些过程中，我们学会了利用博客、书籍等等东西增进我们对任务的理解。

所有人一共攻克一个问题的感觉很好，感受到了团队的力量。在解决这一困难问题的过程中，我意识到一个人的力量是有限的，我们需要别人的帮助，我们组有好几次陷入了一些困境，我们提出了一些解决问题的方法，但是这些方法对代码和整体结构的改动很大，在这种时候，和其他人的交流常常启发

了我们，让我们顺利的将问题解决。

周林锋心得体会：

通过此次计算机系统课程设计，是我更加扎实的掌握了有关 CPU 结构及设计方面的知识，在设计过程中遇到了一些问题，但经过不断探索，检查与分析问题，最后找到了原因所在，也暴露了我在动手实践方面的欠缺与经验不足。实践出真知，通过亲自动手设计，接线，实现 CPU，使我更加深刻的掌握了所学的知识，不再是纸上谈兵。

在课程设计的过程中，我们发现错误，思考方案，不断改正，不断思考，最终和我的队友一起完成全部关键点测试，这也给我留下来深刻的记忆。从刚开始面对 Verilog 的一窍不通，对五级流水的懵懵懂懂，我们焦虑过，困惑过，但学习一个学科，学习一门知识本身就是一个不断遇到困难，不断去学习，通过不懈努力去解决困难的过程。不会 Verilog 便去学习，不懂 CPU 结构、五级流水，便埋头于《自己的手写 CPU》，面对困难我们没有退缩，去不断探索，发现问题所在，然后一一进行解决，记得当时面对第一个数据相关迟迟难以解决，面对一个个陌生的变量名称无从下手时，我们并没有选择放弃，而是选择不厌其烦的一遍遍调试，虚心向进度更快的同学请教，在一次又一次的失败中不断积累，沉淀，最终完成突破。当我们通过第一个关键点时，大家都长舒了一口气，都情不自禁的露出喜悦的神情，因为我们知道，自己的努力没有白费，付出的汗水终究有了回报。这真的给了我很多的思考，不光是在这次课程设计中，当我们迈入社会，面对未来的工作生活时，我们也会遇到这样那样的困难，甚至身边不会有和你分担压力，攻克艰难的伙伴，但我们只要保持一颗谦卑的心，一个永远学习的态度，面对困难不放弃，细心分析，不断尝试，我们终究会克服困难，收获成功。在这次课程设计过程中，也让我明白了团队的重要性，当我一个人遇到难以克服的困难时，我可能会一蹶不振，可能会退缩，面对一个又一个的问题应接不暇时，我和我的队友们就是彼此坚实的后盾，有不会的问题，我们一起讨论，思维的火花就在讨论中诞生，遇到大量的指令编写，CPU 各个功能部件之间加线改线，时序逻辑难以设计的困难，我们分工合作，各取所长，最终完成了任务。这也让我们彼此更加默契，增进了友谊，让我明白了团队的力量。除此之外，老师和助教学长，以及我的同学朋友们也给了我很多的帮助，以及网络上分享知识的博主、教程，在我困惑无助时为我指点迷津，帮助我，教导我，让我明白，一个人的成功不仅是个人努力的结果，也离不开他人无私的帮助。这次课程设计对我是一次不小的挑战，但也是激励我不断向前的动力，令我受益匪浅。

刘腾浩实验心得体会：

在本次实验的过程中，我对面向项目的学习有所感觉。在以往的学习过程中，我们常常是学了一大堆的东西，然后不知道如何去使用，比如学完了数据库系统概论，掌握了不少的知识和概念，但是我们对概念如何落地，如果做成产品一无所知，又或者，我们是在掌握了实验所需的理论之后才开展的实验，比如我们的编译原理课程，我们先学了很多理论，然后运用所学的知识，我们就把一个编译器做好了。本次实验，我们课堂上学到的知识或者技能对于我们解决实验遇到的问题来说是不够充足的。在我们实验开始的阶段，我们需要大概读一读《自己动手写 CPU》，没有本书的帮助，我们搜集信息、解决问题的速度不知道要慢多少。同时本实验还有一个全新的特点，本实验是由企业设计研发的（龙芯中科技术有限公司），这一点还挺重大的。

我们之中的很多人都觉得这是他们遇到的最大的挑战之一，在巨大的挑战面前，我精进了知识，掌握了技能，这些都会陪伴我们很长的时间。