实验报告成绩：	成绩评定日期：

2022～2023 学年秋季学期

《计算机系统》必修课

课程实验报告

班级：人工智能 2001 班

组长：王小可

组员：李婷玉、雷传澳

报告日期：2022.01.01

目录

小组分工

王小可	负责计数器、解决数据相关问题、与雷传澳同学共同完成基础指令的添加、加载存储指令的实现、编写实验报告
李婷玉	负责跳转指令的添加、与雷传澳同学共同完成乘除法部分、编写实验报告。
雷传澳	与王小可同学共同完成基础指令的添加，与李婷玉同学共同完成乘除指令的添加、编写实验报告

总体设计

我们的 CPU 中包含组合逻辑电路和时序逻辑电路，其输入的、运算的、存储的、输出的数据都在这次两种逻辑电路上流转。同时,由于数据通路中会有多路选择器、时序逻辑器件,因此有相应的控制信号,产生这些控制信号的逻辑称为控制逻辑。

我们的 cpu 主要由七个模块组成： IF 、ID 、EX 、MEM 、WB 、CTRL 、regfile ，其中主要修改完善 ID 、EX 、MEM 三个模块通过 64 号测试点。各个模块的主要功能为：

IF:取值阶段，主要功能是将指令取回，并确定下一条指令；

ID:译码阶段，主要功能是解析指令生成控制信号并调用 regfile 模块从通用寄存器读取要使用的通用寄存器的值生成源操作数、以及控制 hilo 寄存器的读取；

EX:执行阶段，主要功能是对源操作数进行算术逻辑类指令的运算，如果是访存指令访存则还要进行指令的地址计算，并设置与内存数据交互的方式；

MEM:访存阶段，主要功能是取回访存的结果；

WB:回写阶段，主要功能是将结果写人通用寄存器堆；

CTRL:控制模块，主要功能是控制流水线是否需要暂停、 冲洗缓存等。

连线图

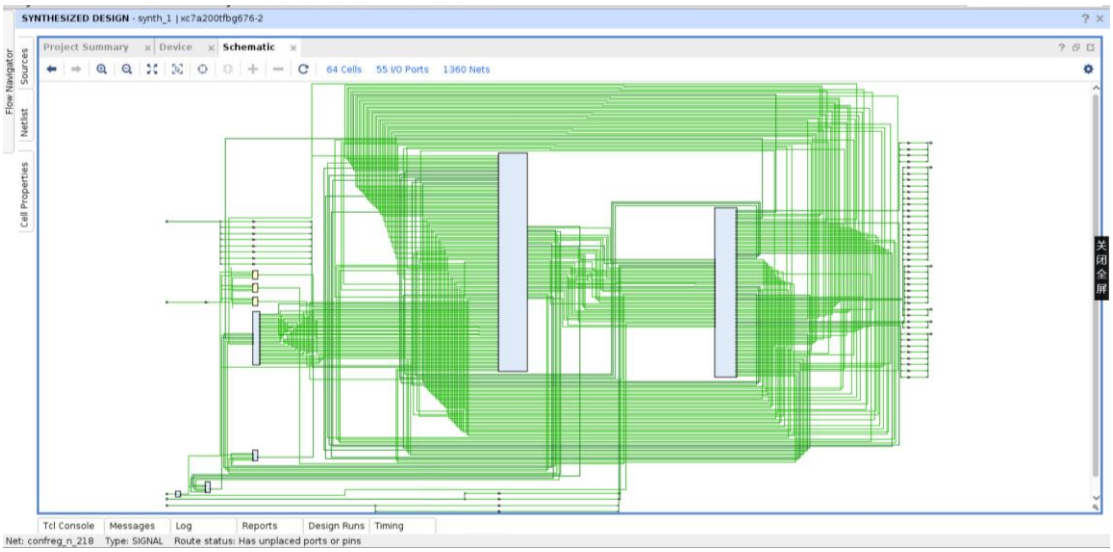

图 1 CPU 连线图

共计完成 58 条指令：

```
wire inst_ori, inst_lui, inst_addiu, inst_beq;
wire inst_subu, inst_jal, inst_jr, inst_addu;
wire inst_sll, inst_or, inst_lw, inst_xor,inst_sb, inst_sh, inst_sw;
wire inst_sltu,  inst_slt;
wire inst_slti, inst_sltiu;
wire inst_j, inst_bne;
wire inst_add, inst_addi;
wire inst_sub, inst_divu;
wire inst_nor, inst_xori;
wire inst_lh, inst_lb, inst_lbu, inst_lhu;
//li++
wire inst_bgezal;
wire inst_bgez, inst_bgtz;
wire inst_bltz, inst_bltzal;
wire inst_jalr;
wire inst_blez;

//lei++
wire inst_and, inst_andi;
wire inst_sllv;
wire inst_sra,   inst_srav,   inst_srl,    inst_srlv;
wire inst_div;  //1.8+
wire inst_mfhi,  inst_mflo,   inst_mthi,   inst_mtlo;//1.8+
wire inst_mult,  inst_multu, inst_mul;//1.9
li++
wire inst_eret,  inst_mfc0,   inst_mtc0;
// wire inst_sb, inst_sh, inst_sw;
wire inst_break, inst_syscall;
```

图 2 添加的指令

程序运行环境及使用工具

程序运行的环境：Vivado 2019.2

使用的工具：VSCode 以及 Verilog 相关的插件，使用 github 完成小组协同工作

单个流水段说明

IF

接口介绍

表格 1 IF 段接口

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制流水线是否暂停
4	flush	1	输入	是否冲洗缓存区
5	new_pc	32	输入	下一条指令指令地址
6	br_bus	33	输入	ID 段发出的分支跳转指令的信号，控制指令延迟槽是否跳转
7	if_to_id_bus	33	输出	IF 段传递 ID 段的系列数据
8	inst_sram_en	1	输出	指令寄存器的读写使能信号
9	inst_sram_wen	4	输出	指令寄存器的写使能信号
10	inst_sram_addr	32	输出	指令寄存器的地址，用来寻找指令的存放的位置
11	inst_sram_wdata	32	输出	指令寄存器的数据，用来存放数据

功能介绍

IF 段的主要功能为控制 pc 值，并根据 pc 值从 sram 中读取指令传到 ID 段中。其中 stall 信号用于控制流水线是否需要暂停；br_bus 为跳转指令信号，用于控制是否需要跳转及跳转的地址；在正常情况下，即没有暂停及跳转时。pc_reg 为当前 next_pc 的值，next_pc 的值加 4。

Regfile

接口介绍

Regfile 模块实现了 32 个 32 位通用整数寄存器的读操作和一个寄存器的写操作。Regfile 模块对应的代码位于 lib 文件下的 regfile.v 中，接口描述如下表所示。

表格 2 Regfile **接口**

序号	**接口名**	**宽度（bit)**	**输入/输出**	**作用**
l	clk	1	输入	时钟信号
2	raddr1	5	输入	第一个读寄存器端口要读取的寄存器地址
3	rdata1	32	输出	第一个读寄存器端口输出的寄存器值
4	raddr2	5	输入	第二个读寄存器端口要读取的寄存器地址
5	rdata2	32	输出	第二个读寄存器端口输出的寄存器值
6	we	1	输入	写使能信号
7	waddr	5	输入	要写入的寄存器地址
8	wdata	32	输入	要写入的数据

功能介绍

Regfile 模块可以分为四段进行理解。

（1）第一段：定义的一个二维变量，元素个数是 32 个，每个元素宽度也是 32，相当于是定义了 32 个 32 位寄存器。

（2）第二段：实现了写寄存器操作，当复位信号无效时，在写使能信号 we 有效，且写操作目的寄存器不等于 0 的情况下，可以将写输入数据保存到目的寄存器。要判断目的寄存器不为 0，因为 MIPS32 架构规定$0 的值只能为 0，所以不写入。

（3）第三段：实现一个读寄存器端口，分为以下 5 步判断：

①当复位信号有效时，第一个读寄存器端口的输出始终为 0。

②当复位信号无效时，如果读取的时$0，那么直接给出 0。

③如果第一个读寄存器端口要读取的目的寄存器与要写入的目的寄存器是同一个寄存器，那么直接将要写入的值作为第一个寄存器端口的输出。

④如果上述情况都不能满足，那么给出第一个寄存器端口要读取的目标寄存器地址对应寄存器的值。

⑤第一个读寄存器端口不能使用时，直接输出 0。

（4）第四段：实现了第二个读寄存器端口，过程参照第三段不再赘述。

ID

现在指令已经进入译码阶段，在此阶段将对取到的指令进行译码：给出要进行的运算类型，以及参与运算的操作数。

接口介绍

ID 模块的作用是对指令进行译码，得到最终运算的类型、子类型、源操作数 1、源操作数 2、要写入的目的寄存器等信息，其中运算类型指的是逻辑运算、移位运算、算术运算等，子类型指的是更加详细的运算类型，比如：当运算类型是逻辑运算时，运算子类型可以是逻辑“或”运算、逻辑“与”运算、逻辑“异或”运算等，ID 模块对应代码在 ID.v 中，接口描述如下表:

表格 3 ID 段接口

序号	接口名	宽度（bit)	输入/输出	作用
l	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	flush	8	输入	是否冲洗缓存区
4	stall	6	输入	是否插入气泡
5	stallreq	1	输出	暂停请求
6	inst_sram_rdata	32	输入	译码阶段的指令
7	ex_we	1	输入	处于执行阶段的指令是否要写目的寄存器
8	ex_waddr	5	输入	处于执行阶段的指令要写的目的寄存器地址
9	ex_wdata	32	输入	处于执行阶段的指令要写入目的寄存器的数据
10	mem_we	1	输入	处于访存阶段的指令是否要写目的寄存器
11	mem_waddr	5	输入	处于访存阶段的指令要写的目的寄存器地址
12	mem_wdata	32	输入	处于访存阶段的指令要写入目的寄存器的数据
13	stall_for_load	1	输出	访存的暂停请求
14	wb_to_rf_bus	38	输入	写回阶段传入寄存器的系列结果
15	id_to_ex_bus	173	输出	译码阶段阶段传入执行阶段的系列结果
16	br_bus	33	输出	控制跳转的总线

功能介绍

ID 段整体功能 ：

ID 段主要功能是解析当前指令，并将当前指令执行需要的资源，选择器或者数据进行预处理，从 regfile 中读出 rs,rt 对应寄存器的值，一并传入 ex 段中，并进行跳转指令的跳转部分的执行。

ID 段功能模块：

I.跳转指令译码

br_e 是一位宽，代表该条指令是否是跳转指令，如果需要跳转，br_e 被赋值为 1'b1, 如果不需要跳转，则 br_e 被赋值为 1'b0。rs_ge_z 也是一位宽，代表是否满足 rdata1_fd 的值大于等于 0，如果满足，会被赋值为 1'b1,如果不能满足，那么它将被赋值为 1'b0 。rs_gt_z 为一位宽，代表是否满足 rdata1_fd 的值大于 0，如果满足，则被赋值为 1'b1, 如果不满足，则被赋值为 1'b0。rs_le_z 是一位宽，代表是否满足 rdata1_fd 的值小于 0，如果满足，则被赋值为 1'b1,如果不满足，则被赋值为 1'b0。rs_lt_z 是一位宽，代表是否满足 rdata1_fd 的值小于 0，如果满足，则被赋值为 1'b1, 如果不满足，则被赋值为 1'b0。rs_eq_rt 是一位宽，代表是否满足 rdata1_fd 是否等于 radta2 的值，如果满足，则被赋值为 1'b1, 如果不满足，则被赋值为 1’b0。br_addr 是三十二位宽，代表跳转后的地址。①beq 指令，将该分支指令对应的延迟槽指令的 PC 加上立即数 offset 左移两位并进行有符号扩展的值赋值给 br_addr。②bne 指令，将立即数 offset 左移 2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算结果赋值给 br_addr。③bgez 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。④bltzal 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。⑤bgtz 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。⑥blez 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。⑦bltz 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。⑧bgezal 指令，将立即数 offset 左移两位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到的值赋值给 br_addr。⑨jal 指令，就将该分支指令对应的延迟槽指令的 PC 的最高四位与立即数 inst[25:0]左移两位后的值拼接之后赋值给 br_addr。⑩j 指令，将该分支指令对应的延迟槽指令的 PC 的最高四位与立即数 inst[25:0]左移两位后的值拼接之后赋值给 bre_addr。

II.设置操作数来源

sel_alu_src1 为三位宽，表示第一个操作数有三种来源:①第一个操作数的值为 rs 寄存器的值②当前的 PC 值③立即数零扩展。sel_alu_src2 为四位宽，表示对于第二个操作数有四种来源：①第二个操作数的值为 rs 寄存器的值②rs 的值为立即数符号扩展③将第二个操作数复制为 32'b8，只有少数指令可以用的到④第二个操作数的值为立即数零扩展。

EX

接口介绍

EX 模块功能:计算 ALU 的结果，根据当前指令，确定即将要写入内存的数据以及地址，或者下一步是否要从内存中读取值。

表格 4 EX 段接口

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	控制暂停信号
4	id_to_ex_bus	169	输入	ID 段传给 EX 段的数据
5	ex_to_mem_bus	80	输出	EX 段传给 MEM 短的数据
6	data_sram_en	1	输出	内存数据的读写使能信号
7	data_sram_wen	4	输出	内存数据的写使能信号
8	data_sram_addr	32	输出	内存数据存放的地址
9	ex_to_id	38	输出	EX 段传给 ID 段的数据
10	data_sram_wdata	32	输出	要写入内存的数据
11	stallreq_from_ex	1	输出	E 发出的是否暂停的信号
12	ex_is_load	1	输出	E 段发给 ID 段的数据，用来判断上一条指令是否是储存指令
13	hilo_ex_to_id	66	输出	EX 段将乘除法器的结果发给 ID 段的 regfile 模块

功能介绍

变量定义：

ex_pc，inst，alu_op，mem_op，sel_alu_src1，sel_alu_src2，data_ram_en，data_ram_wen，rf_we，rf_waddr，sel_rf_res，rf_rdata1，rf_rdata2，is_in_delayslot，hilo_op，imm_sign_extend，imm_zero_extend，sa_zero_extend，alu_src1，alu_src2，alu_result，ex_result，hilo_result，hilo_bus，inst_sb，inst_sh，inst_sw，inst_mfhi，inst_mflo，inst_mthi，inst_mtlo，inst_mult，inst_multu，inst_div，inst_divu，inst_mul，mul_result，mul_signed，div_result，div_ready_i，div_opdata1_o，div_opdata2_o，div_start_o，signed_div_o，hi_we，lo_we，hi_result，lo_result，op_mul，op_div。

变量赋值：

将 ID 段传给 EX 段 id_to_ex_bus_r 赋值给相应变量。判断当前指令是否为 LW 指令，即 inst[31：26]是否 6'b10_0011，如果是那么就将 ex_is_load 赋值为 1;如果不是，就将 ex_is_load 赋值为 0。计算 ALU 来个操作数的值：定义立即数符号扩展的变量并将其赋值为{ {16{inst[15]}}， inst[15：0]}，定义立即数零扩展的变量并将其赋值为{16'b0， inst[15：0]}，定义 s a 零扩展的变量并将其赋值为{27'bo，inst[10：6]}计算参与 ALU 运算的操作数，根据 ID 段，传过来的操作数来源的方式，也就是 sel_alu_src1，sel_alu_src2 中的值，然后给 alu_src1，alu_src2 赋值成对应的值。调用 ALU 模块，将参与 ALU 计算的两个操作数，已经 ALU 计算的方式传给 ALU 的接口，然后从 ALU 模块中得到 ALU 计算的结果，将其赋值给 ex_result。

读写内存：

将内存读写使能设置为相应的值。判断当前指令是否是 sb 指令，即 data_ram_readen 是否为 4'b0101，如果是，判断要写入内存的值，即判断要写入内存地址的后两位的值为多少，如果 ex_result[1：0] == 2'b00，那么就将内存写使能赋值为 4'b0001，表示要写入的是第一个字节，如果 ex_result[1：0] == 2'b01，那么就将内存写使能赋值为 4'b0010，表示要写入的是第二个字节，如果 ex_result[1：0] == 2'b10，那么就将内存写使能赋值为 4'b0100，表示要写入的是第三个字节，如果 ex_result[1：0] == 2'b11，那么就将内存写使能赋值为 4'b1000，表示要写入的是第四个字节;如果不是 sb 指令，那么判断是否是 sh 指令，即 data_ram_readen 是否 4'b0111，如果是，判断要写入内存的值，即判断要写入内存地址的后两位的值为多少如果

读内存：将 data_sram_en 赋值为对应的值后，1 表示要可以读内存，0 表示不可以读取内存，此外将 data_sram_addr 赋值为要读内存的地址，在 EX 会获取到想要读取内存地址的数据。

发送 EX 段结果给其他段：发给 B 段：将内存的读使能信号，当前的 Pc 值，内存的读写使能，以及内存写使能信号，以及寄存器的写使能信号，以及寄存器要写的地址与数据。发给 ID 段。寄存器的写使能信号，以及寄存器要写的地址与数据，用来让 ID 段判断是否会出现相关的情况发生。还有乘除法器高位寄存器以及低位寄存器的写使能信号，以及乘除法器高位和低位要写入的数据，让 ID 段在调用 regfile 的同时，将乘除法器高位和低位的值也一并写入寄存器中，提高了 CPU 效率。

乘除法指令的实现：

在 EX 段我们加入了 MUL 和 DIV 模块，借此完成乘法和除法的指令。在收到来自 ID 段的 id_to_ex_bus 后，可以判别是否为乘数法，并调用相应的乘法或除法模块进行运算。

乘法指令的实现：

如果是乘法，则需要声明几个变量，分别为：stallreq_for_mul、mul_ready_i、signed_mul_o、mul_opdatal_o、mul_opdata2_o、mul_start_o。stallreq_for_mul 表明是否因为多周期的 mul 进行流水线的暂停，mul_ready_i 表明乘法是否已经结束，signed_mul_o 表示是否为有符号的乘法，mul_opdatal_o、mul_opdata2_o 是 ID 段传过来的两个操作数，

mul_start_o 表明是否开始乘法运算。具体的乘法器模块的解释说明在下文中给出，这里需要说明的是对传入乘法器 mymul 的相应的值的赋值。

当检测到乘法的指令 inst_mult 或者 inst_multu 乘法器就会开始运作，开始前，会先判断当前乘法器的状态，若当前乘法器为空闲的状态，则将操作数传入乘法器开始运算，同时因为该乘法器是 32 周期的，因此需要对流水线进行暂停，将 stallreq_for_mul 的值改为`Stop 并传给 CTRL 模块进行流水线暂停。当乘法指令运行结束后，stallreq_for_mul 的值改为`NoStop，流水线继续运行。乘法的运行结果存放到了 mul_result 中。

除法指令的实现：

除法指令的实现与乘法类似，都是先判断是否为有符号除法或无符号除法，然后判断除法器的状态是否为空闲，除法器可用时，就把被除数和除数传入除法器，并把 stallreq_for_div 值改为`Stop 来暂停流水线。在除法结束后结果放在 div_result 中并恢复流水线继续运行。

在乘除法结束后，需要将结果存到 hilo 寄存器中。首先把 result 中的高 32 位的值和低 32 位的值放在对应的 hi、lo 的值中，然后把要写入的值和写使能信号直接发送给 ID 段。因为在 MEM 和 WB 段没有涉及到 hilo 寄存器的读写的问题，因此我们直接把 hilo 的写入的线发给了 ID 段，并没有像通用寄存器一样向后传到 WB 段再发给 ID 段。至此，乘法和除法的指令已经可以正常运行。

MEM

接口介绍

表格 5 MEM 段接口

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	flush	1	输入	是否冲洗缓存区
4	stall	6	输入	控制流水线是否暂停
5	ex_to_mem_bus	147	输入	执行阶段传入的系列结果
6	mem_to_wb_bus	136	输出	回写阶段需要接收的系列结果
7	data_sram_rdata	32	输入	存取指令需要访问的数据存储器的数据

功能介绍

在访存阶段将该阶段的处理结果在下一个时钟周期传递到回写阶段，如果是访存指令，则还需要访问数据存储器。在访问数据存储器时。使用的是时序逻辑，即在时钟上升沿才发生信号传递。

WB

接口介绍

表格 6 WB 段接口

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号

3	flush	1	输入	控制是否冲洗缓冲区
4	stall	1	输入	控制流水线是否暂停
5	mem_to_wb_bus	136	输入	访存阶段传递给回写阶段的系列数据
6	wb_to_rf_bus	38	输出	回写阶段传递给 rf 模块的系列数据
7	hilo_bus	66	输出	传入 hilo_reg 的写入请求合并
8	debug_wb_pc	32	输出	用来调试的 pc 值
9	debug_wb_rf_wen	4	输出	用来调试的写使能信号
10	debug_wb_rf_wnum	5	输出	用来调试的寄存器地址
11	debug_wb_rf_wdata	32	输出	用来调试的寄存器数据

功能介绍

在回写阶段接收前面阶段的处理结果，这个阶段的实际内容主要是通过 Regfile 模块实现的，将 rf_we、rf_waddr、rf_wdata 传入到 regfile 模块，分别连接到 regfile 模块的写使能端口 we、写操作目的寄存器端口 waddr、写入数据端口 wdata，所以会将指令的运算结果写入目的寄存器。

```
assign wb_to_rf_bus = {
    rf_we,          // 37
    rf_waddr,       // 36:32
    rf_wdata        // 31:0
};

assign debug_wb_pc = wb_pc;
assign debug_wb_rf_wen = {4{rf_we}};
assign debug_wb_rf_wnum = rf_waddr;
assign debug_wb_rf_wdata = rf_wdata;
```

图 3 WB 模块部分代码

Forwarding 通路

存在的问题

数据相关指的是在流水线中执行的几条指令中，一条指令依赖于前面指令的执行结果，一共有三种情况:RAW、WAR、WAW。

因为只在流水线回写阶段才会写寄存器，因此不存在 WAW 相关，又因为只能在流水线译码阶段读寄存器、回写阶段写寄存器，因此不存在 WAR 相关，所以我们设计的 CPU 中只存在 RAW 相关。RAW 相关有三种情况。

①相邻指令间存在数据相关

②相隔 1 条指令的指令间存在数据相关

③相隔 2 条指令的指令间存在数据相关

解决办法

对于相隔两条指令的数据相关问题，在读操作中作一个判断，如果要读取的寄存器是在下一个时钟上升沿要写入的寄存器,那么就将要写入的数据直接作为结果输出；对于相邻指令间存在数据相关、相隔 1 条指令的指令间存在数据相关这两种情况，有三种解决方法。

①插入暂停周期：当检测到相关时，在流水线中插入一些暂停周期，

②编译器调度：编译器检测到相关后，可以改变部分指令的执行顺序

③数据前推：将计算结果从其产生处直接送到其他指令需要处或所有需要的功能单元处，避免流水线暂停

我们在设计时，使用了 forwarding 技术来解决数据相关问题。如果需要的源寄存器就是在上一个周期的执行阶段（或者访存阶段）要写的目的寄存器，则直接把执行阶段（或访存阶段）的结果作为源寄存器的值;

```
assign rs_ex_ok =  (rs == ex_waddr)  &&  ex_we ? 1'b1 : 1'b0;  //ex_we
assign rt_ex_ok =  (rt == ex_waddr)  &&  ex_we ? 1'b1 : 1'b0;

assign rs_mem_ok = (rs == mem_waddr) && mem_we ? 1'b1 : 1'b0;
assign rt_mem_ok = (rt == mem_waddr) && mem_we ? 1'b1 : 1'b0;

assign sel_rs_forward = rs_ex_ok | rs_mem_ok;  //forwarding
assign sel_rt_forward = rt_ex_ok | rt_mem_ok;  //forwarding

assign rs_forward_data = rs_ex_ok  ? ex_wdata :  //data
                         rs_mem_ok ? mem_wdata:
                         32'b0;

assign rt_forward_data = rt_ex_ok  ? ex_wdata :  //data
                         rt_mem_ok ? mem_wdata:
                         32'b0;

assign rdata1_fd = sel_rs_forward ? rs_forward_data : rf_rdata1;
assign rdata2_fd = sel_rt_forward ? rt_forward_data : rf_rdata2;

assign stall_for_load = ex_ram_read & (rs_ex_ok | rt_ex_ok);  //+}
```

图 4 Forwarding 技术核心部分

加载存储指令

共有 14 条指令:lb、lbu、lh、lhu、ll、lw、lwl、lwr、sb、sc、sh、sw、swl、swr，以“l”开始的都是加载指令，以“s”开始的都是存储指令，这些指令的主要功能为从存储器中读取数据，或者向存储器中保存数据。因为我们的存储器是使用的字节编址，所以 data_sram_wen 这个写使能信号是 4 位，每一位控制一个字节的数据。所以 sw 指令对应 4'b1111，lw 指令对应 4'b0000，其他的存储指令同理。

加载指令 lb、lbu、lh、lhu、lw

加载指令在译码阶段进行译码，得到运算类型 alusel_o、aluop_o，以及要写的目的寄存器信息。这些信息传递到执行阶段，然后又传递到访存阶段，访存阶段依据这些信息，设置对数据存储器 RAM 的访问信号。从 RAM 读取回来的数据需要按照加载指令的类型、加载地址进行对齐调整，调整后的结果作为最终要写入目的寄存器的数据。

31 26 25 21 20 16 15 0

LB 100000	base	rt	offset
LBU 100100	base	rt	offset
LH 100001	base	rt	offset
LHU 100101	base	rt	offset
LW 100011	base	rt	offset

图 5 加载指令 lb、lbu、lh、lhu、lw 的格式

从图 5 可知，这 5 条加载指令可以根据指令中 26-31bit 的指令码加以区分，另外，加载指令的第 0～15bit 是 offset、第 21~15bit 是 base，加载地址的计算方法如下，先将 16 位的 offset 符号扩展至 32 位，然后与地址为 base 的通用寄存器的值相加，即可得到加载地址。

由于这个存储器只配置了片选(4byte)使能和字节写使能，所以读取的时候一律是先读回 CPU(此时不区分是哪个 load 指令)，在 MEM 段再进行更细分的操作。

存储指令 sb、sh、sw

存储指令在译码阶段进行译码，得到运算类型 alusel_o、aluop_o，以及要存储的数据。这些信息传递到执行阶段，然后又传递到访存阶段，访存阶段依据这些信息，设置对数据存储器 RAM 的访问信号，将数据写入 RAM。

31 26 25 21 20 16 15 0

SB 101000	base	rt	offset
SH 101001	base	rt	offset
SW 101011	base	rt	offset

图 6 存储指令格式

从图 6 可知，这 3 条存储指令可以根据指令中 26~31bit 的指令码加以区分，另外，存储指令的第 0~15bit 是 offset、第 21～15bit 是 base，存储地址的计算方法如下，先将 16 位的 offset 符号扩展至 32 位，然后与地址为 base 的通用寄存器的值相加，即可得到存储地址。

store 类指令与 load 类指令略有不同，sb 根据写地址的最低两位 addr[1:0]判断是 4'b0001、 4'b0010、 4'b0100、 4'b1000 中的某种情况；sh 只有两种情况，地址最低两位为 00 对应低位两个字节，即 data_sram_wen 为 4'b0011；地址最低两位为 10 时对应高位两个字节，即 data_sram_wen 为 4'b1100。

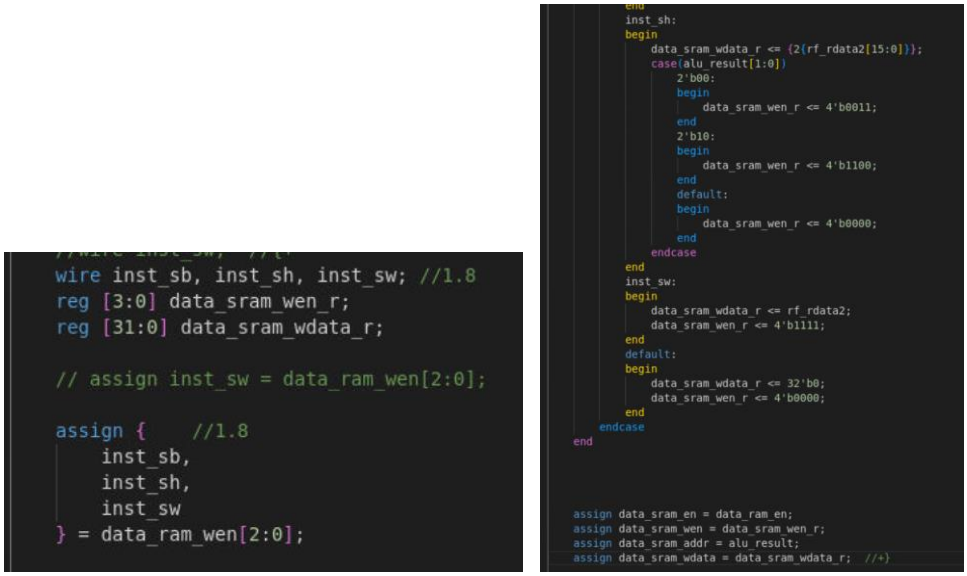

图 7 部分 store 代码

实验心得

王小可

CPU 的设计实验开始之前于亚新老师通过线上课堂的形式为我们讲授了理论知识，但是由于知识停留在了浅层及基础知识，对于如何实际上手却并不是特别明白，于是通过 CPU 设计实验加深我们对计算机的指令处理机制、数据通路的解决技术（forwarding 技术、加气泡技术）的认识与理解。在刚开始拿到实验的时候，对 verilog 语言、vivado 的使用都十分陌生，在学长的耐心指导与教程讲解下才逐渐开始上手，特别是刚开始写模 10 计数器时，因为不太懂 verilog 语言所以犯了挺多错误，但是通过不断查阅资料、寻求学长帮助，最后完成了一个基础的模 10 计数器，对后面上手写 CPU 提供了不少帮助。代码修改过跟多遍，每一遍都很心酸，但是更加熟悉了各个模块的作用，使用的原理。单独模块其实挺简单的，根据数据通路图就可以写出来，但是如何将整个流水段连接起来需要我们多加思考。

最后也非常感谢于亚新老师和孙平炜学长的悉心教导、能够真正替我们着想，为我们能够掌握计算机系统基础知识花费了不少心血，虽然很遗憾的是由于疫情原因没能参加老师的线下课程，课堂的效率有所影响，但是疫情终将过去，期待我们下学期相约线下课堂。

李婷玉

在本次的计算机系统实验课上我收获了很多，对基础知识有了实践层面的认识。还记得第一次刚刚开始完成实验的时候，我们小组都是比较懵的状态，感觉自己的知识学习的很浅，后来通过一次次的腾讯会议讨论、学习，我们小组开始逐渐的理解实验的要求开始逐渐上手，最终完成了本次的实验，夯实了基础，提高了自己动手的能力，当然在这期间也很感谢学长以及老师提供的帮助，学长对我们的问题解释的很清楚，会分析我们的错误点。当然本次的实验我也感受到了团队协作的重要性，当一个小组需要完成一个工程需要每一位成员的努力和理解，合作一起推进。

雷传澳

非常感谢于亚新老师和孙平炜学长对我们的指导，简直帮助太大了，也非常感谢队友王小可、李婷玉一路上的包容与支持，一起合作推进完成了这个 CPU 的设计实验。

起初，我对 CPU 的了解还停留在非常浅的层面，比如 CPU 是处理数据的地方，当我们遇到信息量大的情况，会说我的 CPU 快烧坏了。但是经过《计算机系

统》《计算机组成原理》的学习，经过老师和学长的耐心讲解，发现自己之前对 CPU 的了解简直是九牛一毛。而从了解 CPU 到明白它的工作原理，再到自己动手做一个 CPU，整个过程可以说是道阻且长，但是也真的能学到很多知识，更真切的明白 CPU 的工作原理，对计算机有更深的一个认识。

参考资料

[1] 雷磊思.自己动手做 CPU[M].北京：电子工业出版社，2014：107-229.

[2] 《CPU 设计实战》——汪文祥、邢金璋

[3] 张晨曦 著《计算机体系结构》（第二版） 高等教育出版社

[4] 《A03_“系统能力培养大赛”MIPS 指令系统规范_v1.01》

[5] 《A07_vivado 使用说明_v1.00》

[6] 《A09_CPU 仿真调试说明_v1.00》

[7] 《A11_Trace 比对机制使用说明_v1.00》