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 2002

组长：魏泽旭

组员：曹宇博，覃伟境

报告日期：2022.12.21

目录

1. 工作量及完成指令：

1.1. 魏泽旭 40%：主要负责数据通路，一号点通过及相应指令添加，数据相关 stall 处理及 forwarding，实现乘除法指令添加。

1.2. 曹宇博 35%：非乘除法运算指令实现，存储/读取指令实现

1.3. 覃伟境 25% 分支指令实现，部分运算指令实现

2. 环境及具体流水段

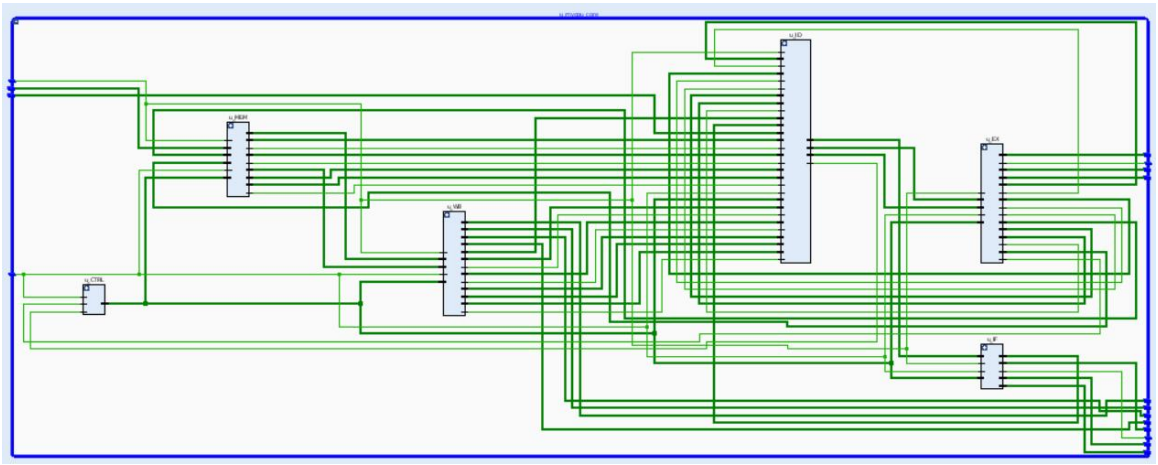
2.1. 实验环境

在实验室 cg 平台上，使用 vivado 进行功能仿真作为程序运行环境，通过 Vscode 编程，使用 github 远程协作。

2.2. 总体架构

总体采用五段流水的结构,IF 段取值、ID 段译码、EX 段有效地址计算、MEM 段访问内存、WB 段 写回寄存器。同时采用 ctrl 段进行流水线的暂停控制，alu 进行指令的运算，refile 进行寄存器实现。

连线图如下

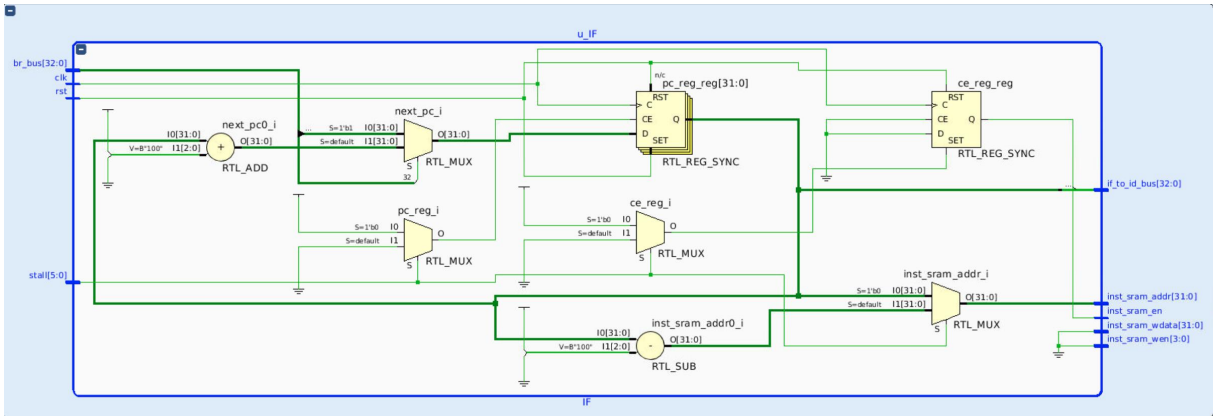


2.3 IF 流水段

2.3.1 功能说明:

IF 段主要进行取指操作，根据当前 PC 寄存器里的 PC 值从内存中取出指令，将指令传到 ID 段进行译码操作。

2.3.2 信号说明:



序号	名称	位宽	I/O	功能
1	br_bus	33	Input	跳转总线
2	clk	1	Input	时钟信号
3	rst	1	Input	复位信号
4	stall	6	Input	流水线暂停信号
5	if_to_id_bus	33	Output	IF 段传到 ID 总线
6	inst_sram__addr	32	Output	内存访问地址
7	inst_sram_en	1	Output	内存是否可访问
8	inst_sram_wdata	32	Output	写内存的值
9	inst_sram_wen	4	Output	内存是否可写

将总线 br_bus 拆解为以下信号：

序号	名称	位宽	功能
1	br_e	1	是否跳转
2	br_addr	32	跳转到的地址

2.3.3 功能模块说明

在每个时钟周期开始时，若 IF 段不暂停，则 PC 寄存器存入上一段的 next_pc 值，即本段 PC 值，CE 寄存器赋为 1 表示可取指。

next_pc 赋值如下：

```
assign next_pc = br_e ? br_addr
                 : pc_reg + 32'h4;
```

next_pc 的赋值根据是否跳转来决定，跳转则为跳转到的地址，否则 PC+4。

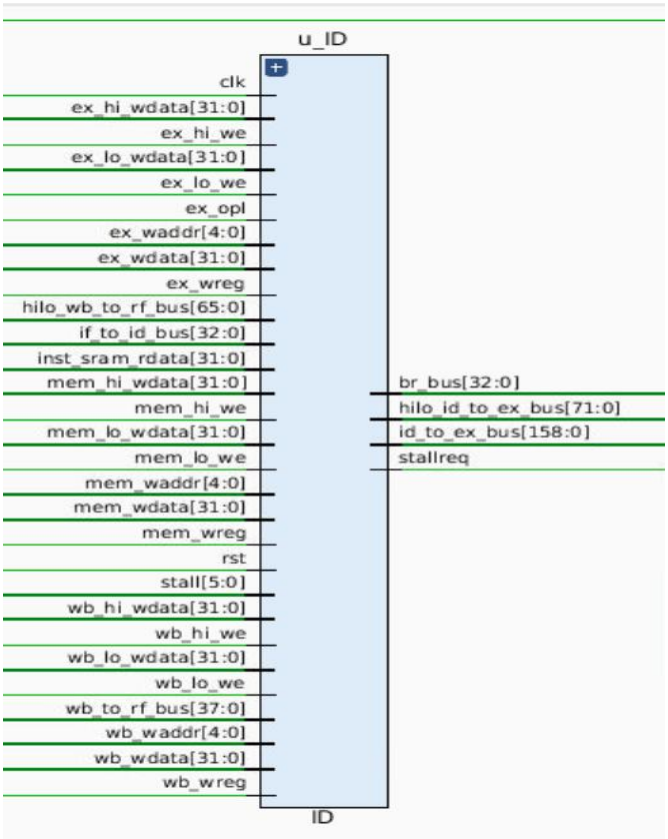
inst_sram_en、inst_sram_wen、inst_sram_addr、inst_sram_wdata、if_to_id_bus 赋值如下：

```
assign inst_sram_en = ce_reg;
assign inst_sram_wen = 4'b0;
assign inst_sram_addr = (stall[0] == `NoStop) ?
pc_reg :pc_reg-32'h4;
assign inst_sram_wdata = 32'b0;
assign if_to_id_bus = {
    ce_reg,
    pc_reg
};
```

inst_sram_en 赋 ce_reg 的值表示是否可访问内存取指，inst_sram_addr 赋 pc_reg 的值表示取指地址。需要说明的是，若 IF 段暂停，pc_reg 内已经是下一条指令的 pc 值，使用 inst_sram_addr 在内存中取出的指令会直接送到 ID 段，因此，inst_sram_addr 应为 ID 段的指令值，即 pc_reg 内地址的上一条指令。因此，在 IF 段暂停时，应该将 pc_reg-4 赋给 inst_sram_addr。

2.4 ID 流水段

2.4.1 信号说明:



序号	名称	位宽	I/O	功能
1	clk	1	Input	时钟信号
2	ex_hi_wdata	32	Input	EX 段新的寄存器数值
3	ex_hi_we	1	Input	EX 段是否传入 HI 寄存器
4	ex_lo_wdata	32	Input	EX 段 LO 寄存器的值
5	ex_lo_we	1	Input	EX 段是否传入 LO 寄存器
6	ex_opl	1	Input	EX 段是否为 load 指令
7	ex_waddr	5	Input	EX 段新的寄存器数值
8	ex_wdata	32	Input	EX 段是否为 load 指令
9	ex_wreg	1	Input	EX 段是否有传入寄存器
10	hilo_wb_to_rf_bus	66	Input	HILO 寄存器的 WB 线
11	if_to_id_bus	33	Input	IF->ID 总线
12	inst_sram_rdata	32	Input	传入的指令机器码

13	mem_hi_wdata	32	Input	MEM 段 HI 寄存器的值
14	mem_hi_we	1	Input	MEM 段是否传入 HI 寄存器
15	mem_lo_wdata	32	Input	MEM 段 LO 寄存器的值
16	mem_lo_we	1	Input	MEM 段是否传入 LO 寄存器
17	mem_waddr	5	Input	MEM 段传入寄存器地址
18	mem_wdata	32	Input	MEM 段新的寄存器数值
19	mem_wreg	1	Input	MEM 段是否有传入寄存器
20	rst	1	Input	复位信号
21	stall	6	Input	输入的延迟信号
22	wb_hi_wdata	32	Input	WB 段 HI 寄存器的值
23	wb_hi_we	1	Input	WB 段是否传入 HI 寄存器
24	wb_lo_wdata	32	Input	WB 段 LO 寄存器的值
25	wb_lo_we	1	Input	WB 段是否传入 LO 寄存器
26	wb_to_rf_bus	38	Input	WB->寄存器总线
27	wb_waddr	5	Input	WB 段传入寄存器地址
28	wb_wdata	32	Input	WB 段新的寄存器数值
29	wb_wreg	1	Input	WB 段是否有传入寄存器
30	br_bus	33	Output	地址线
31	hilo_id_to_ex_bus	72	Output	HILO 寄存器的 ID->EX 线
32	id_to_ex_bus	159	Output	ID->EX 总线
33	strallreq	1	Output	输出的延迟信号

以 ex、mem、wb 开头的就是用于局部回路的组线。其中，结尾为 wreg 和 we 的提示是否存在回路的回调数据，结尾为 waddr 的是传回寄存器的号数，而结尾为 wdata 的则是传回的数据本身。如果 waddr 正好和要调用的 rs 或 rt 相同，同时 wreg 处于 1 状态（有回调数据），则用 wdata 取代自寄存器组件中读出的数据。对于 hi 和 lo 两个寄存器同理，只起到存在性作用的换成 hi/lo_we 结尾的导线，而传回的数据则是 hi/lo_wdata 结尾的导线组。

2.4.2 功能模块说明：

运行机制：

ID 段接入总线有四条：inst_sram_rdata、wb_to_rf_bus、if_to_id_bus 和 hilo_wb_to_rf_bus，输出有三条： id_to_ex_bus、br_bus 和 hilo_id_to_ex_bus:

以下涉及到 hi 和 lo 的会被并入 hilo_id_to_ex_bus 总线：其余的均会被并入 id_to_ex_bus 总线，两者均传到 EX 段。

1. inst_sram_rdata 为按时序逻辑依次读入的指令机器码，其会被按位拆解为数个不同的部分，如最高六位为 op_code，经译码器解析后归类为一每个指令独有的导线单位：

```
assign inst_ori     = op_d[6'b00_1101];
assign inst_lui     = op_d[6'b00_1111];
assign inst_addiu   = op_d[6'b00_1001];
assign inst_beq     = op_d[6'b00_0100];
assign inst_bne     = op_d[6'b00_0101];
```

有些指令需要最低六位 func 经过译码器解析后辅助判定其类别：

```
assign inst_sra     = op_d[6'b00_0000] & func_d[6'b00_0011];
assign inst_srav    = op_d[6'b00_0000] & func_d[6'b00_0111];
assign inst_srl     = op_d[6'b00_0000] & func_d[6'b00_0010];
assign inst_srlv    = op_d[6'b00_0000] & func_d[6'b00_0110];
assign inst_nor     = op_d[6'b00_0000] & func_d[6'b10_0111];
```

接下来的部分大多都与其属于何种指令（哪个 inst_导通）有着密不可分的关系。比如确定 EX 段 alu 要执行哪个装置：

```
assign op_add = inst_addiu | inst_lsa | inst_jal | inst_addu | inst_lw
| inst_sw|inst_lb | inst_lbu | inst_lh |inst_lhu | inst_sb | inst_sh
| inst_add | inst_addi | inst_bltzal | inst_bgezal | inst_jalr ;
assign op_sub = inst_subu | inst_sub;
assign op_slt = inst_slt | inst_slti;
assign op_sltu = inst_sltu | inst_sltiu;
assign op_and = inst_and |inst_andi ;
assign op_nor = inst_nor ;
assign op_or = inst_ori | inst_or;
assign op_xor = inst_xor| inst_xori ;
assign op_sll = inst_sll| inst_sllv ;
assign op_srl = inst_srl | inst_srlv ;
assign op_sra = inst_sra | inst_srav ;
assign op_lui = inst_lui;

assign alu_op = {op_add, op_sub, op_slt, op_sltu,
                 op_and, op_nor, op_or, op_xor,
                 op_sll, op_srl, op_sra, op_lui};
```

确定两个操作数，通过 SEL_ALU_SRC1 和 SEL_ALU_SRC2 判断，两个导线组采用独热编码：

SEL_ALU_SRC1：由低到高三位，分别代表第一个操作数取为 rs 段号的寄存器，取 pc 值和取 sa 段数值的无符号拓展；

SEL_ALU_SRC2：有四位，代表第二操作数的取法：由高到低分别是取 rt 段操作数的值，取立即数区的符号拓展，取正整数 8 和取立即数区的无符号扩展。

相似的，还有 sel_rf_dst 这一负责指示结果储存寄存器的导线组，从低到高分别表示 rd 段号寄存器，rt 段号寄存器以及 31 号寄存器，然后将运算出的内容写入 rf_waddr 中：

```
assign rf_waddr = {5{sel_rf_dst[0]}} & rd
                | {5{sel_rf_dst[1]}} & rt
                | {5{sel_rf_dst[2]}} & 32'd31;
```

读写方面，inst_X 决定了 data_ram_en 和 data_ram_wen，两者分别是“能否读写内存”和“读写内存使能信号”，负责寄存器和内存的数据交换管理：

```
// load and store enable
assign data_ram_en = inst_lw | inst_sw | inst_lb | inst_lbu | inst_lh
| inst_lhu | inst_sb | inst_sh;

// write enable

assign data_ram_wen = inst_sw ? 4'b1111 :
                  inst_lw ? 4'b0000 :
                  inst_lb ? 4'b1110 :
                  inst_lbu? 4'b1101 :
                  inst_lh ? 4'b1100 :
                  inst_lhu? 4'b0110 :
                  inst_sb ? 4'b0001 :
                  inst_sh ? 4'b0011 : 4'b0;
```

2. 由上可知，作为操作数的寄存器只可取 rs 段和 rt 段，所以此处采用每次运行都调用一次 regfile 读出 rs 段和 rt 段寄存器的内容，无论其是否被用到，读出的内容都将被存放在 RDATA1 和 RDATA2 中。同时，regfile 模块也负责读入 hi/lo 寄存器的内容存入 hi/lo_rdata，并将从 MEM、EX 和 WB 三段传来的数组信号调入 regfile，从而实现局部回路功能。

3. sel_rf_res 段标记 EX 段 alu 的计算结果是要存到目标的直接数据还是某个要读取的内存地址。rf_we 代表指令是否需要读写普通寄存器。lo/hi_e 代表指令是否读 hi 和 lo 两个寄存器。lo/hi_rf_we 代表指令是否会写入 hi 或 lo 两个寄存器。

除以上三个信号外其余信号均会被并入 id_to_ex_bus 总线，并和 hilo_id_to_ex_bus 一起传到 EX 段。

if_to_id_bus 是由 IF 段传入的总线，携带有当前的 pc 值信息，供 ID 段某些操作调用并传导到 EX 段。

wb_to_rf_bus 和 hilo_wb_to_rf_bus 是两条从上一个 WB 段传来的总线，两条总线进行如下拆分：

```
assign {
    wb_rf_we,
    wb_rf_waddr,
    wb_rf_wdata
} = wb_to_rf_bus;
assign {
    wb_hi_rf_wdata,
    wb_lo_rf_wdata,
    wb_rf_hi_we,
    wb_rf_lo_we
} = hilo_wb_to_rf_bus;
```

之后将其均接入 regfile 模组，用于进行 regfile 模组中各寄存器内容的更新，实现写寄存器操作。其中和局部回路部分类似，we 结尾代表对应的单元是否有变动（wb_rf_we 代表标准寄存器，wb_rf_hi/lo_we 代表 hi 和 lo 寄存器），而 wb_rf_waddr 代表要写入的地址，wdata 结尾则是要写入各对应寄存器的具体数值。

br_bus 线传到下一个指令的 IF 段，代表是否发生跳转。bf_bus 的构造为：

```
assign br_bus={
    br_e,
    br_addr
};
```

其中，br_e 为是否进行跳转，br_addr 为跳转到的目标指令地址。

br_e 的判断由多种要素共同决定。首先，根据指令种类（inst_X），给出几个作为判定容器的导线：

```
assign rs_eq_rt = (rdata1 == rdata2);
assign rs_geq_ze = (rdata1[31] == 1'b0);
assign rs_gt_ze = (rdata1[31] == 1'b0 && rdata1 != 0); //rs>0
assign rs_leq_ze = (rdata1[31] == 1'b1 || rdata1 == 0); // rs<=0
assign rs_lt_ze = (rdata1[31] == 1'b1); //rs<0
```

以下是传输判断 RDATA1 和 RDATA2 是否满足某种条件的导线。它们与对应的指令导线求与即是 br_e 的值：

```
assign br_e = (inst_beq & rs_eq_rt) |(inst_bne & ~rs_eq_rt) | (inst_jal
| inst_jr | inst_j | (inst_bgez & rs_geq_ze) | (inst_bgtz & rs_gt_ze)
| (inst_blez & rs_leq_ze) | (inst_bltz & rs_lt_ze) | (inst_bltzal &
rs_lt_ze) | (inst_bgezal & rs_geq_ze) | inst_jalr;
```

可以发现，有一些指令是不需要这类 rs 导线限制的，因为它们是无条件跳转指令。还有一 些如 inst_bne，其实现是由某个的 rs 类导线求反得到的。而 br_addr 为跳转的目标地址，其实现第一取决于跳转指令，不同指令目标地址的给出格式

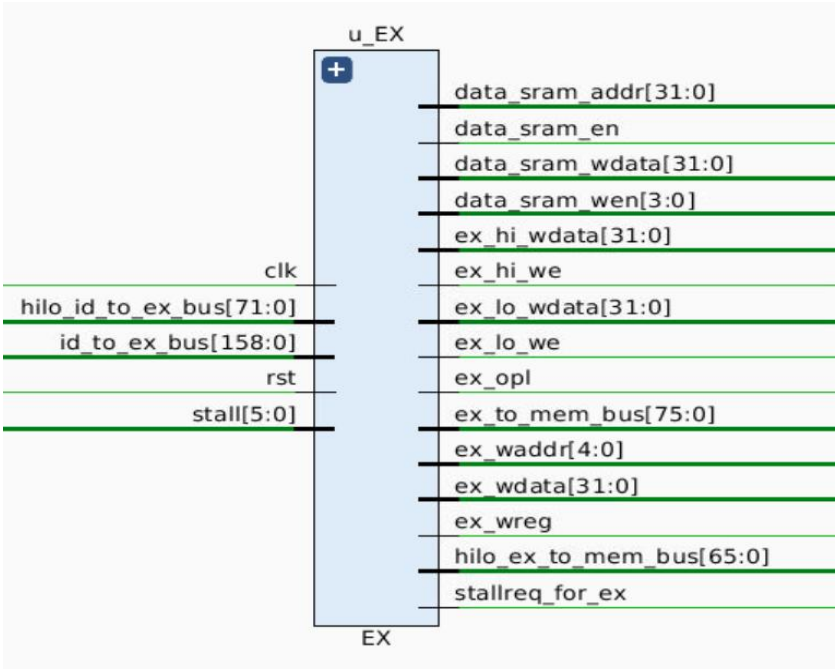
不同。第二取决于操作数：第一部分为 pc_plus_4=id_pc+32'H4，第二个大多数情况下是指令的立即数部分的扩展。通过移位和拼接运算，可以算出 br_addr。

2.5 EX 流水段

2.5.1 功能说明：

EX 段主要进行有效地址计算，其中包含的功能有：对算数运算指令计算出其结果；对 store 指令计算出存入的地址，将要存入内存的值和地址传给 mycore；对 load 指令计算出要取得内存地址传给 mycore；对乘除指令调用 mul 或 div 模块计算出结果传给 hilo 总线。当需要发出暂停时，stallreq_for_ex 置 1 并传给 ctrl，同时将 ex 段的计算结果传给 ID 段构成数据通路，避免数据相关。

2.5.2 信号说明:



序号	接口名	位宽	I/O	功能
1	clk	1	Input	时钟信号
2	rst	1	Input	复位信号
3	stall	6	Input	流水线暂停信号
4	id_to_ex_bus	159	Input	自 ID 段传入 EX 的总线
5	hilo_id_to_ex_bus	72	Input	自 ID 段传入 EX 的乘除总线
6	data_sram_addr	76	Output	EX 段传到 MEM 段的总线
7	data_sram_en	66	Output	EX 段传到 MEM 段的乘除总线

8	data_sram_wen	1	Output	内存是否可访问
9	data_sram_wdata	4	Output	内存写使能信号
10	ex_hi_wdata	32	Output	内存访问地址
11	ex_hi_we	32	Output	写内存的值
12	ex_wreg	1	Output	EX 段写使能信号
13	ex_waddr	5	Output	EX 段写寄存器地址
14	ex_wdata	32	Output	EX 段写寄存器值
15	ex_opl	1	Output	EX 段指令是否是 load 指令
16	ex_hi_we	1	Output	EX 段指令是否写 hi 寄存器
17	ex_lo_we	1	Output	EX 段指令是否写 lo 寄存器
18	ex_hi_wdata	32	Output	EX 段指令写 lo 寄存器值
19	ex_lo_wdata	32	Output	EX 段指令写 lo 寄存器值
20	stallreq_for_ex	1	Output	EX 段是否发出暂停信号

EX 段接入两条总线 id_to_ex_bus 和 hilo_id_to_ex_bus：

id_to_ex_bus：在每个时钟上升沿在不暂停的情况下存入寄存器，并被拆解为以下信号：

序号	名称	位宽	作用
1	ex_pc	32	EX 段 PC 值
2	inst	32	EX 段指令码
3	alu_op	2	算数指令操作码
4	SEL_ALU_SRC1	3	判定操作数 1 选择信号
5	SEL_ALU_SRC2	4	判定操作数 2 选择信号
6	data_ram_en	1	内存是否可访问
7	data_ram_wen	4	对 load 和 store 指令汇总
8	rf_we	1	寄存器是否可写
9	rf_waddr	5	写寄存器地址
0	sel_rf_res	1	选择写入寄存器的值

11	RF_RDATA1	32	rs 寄存器的值
12	RF_RDATA2	32	rt 寄存器的值

hilo_id_to_ex_bus：在每个时钟上升沿在不暂停的情况下存入寄存器，并被拆解为以下信号：

序号	名称	位宽	作用
1	hilo_inst	4	hilo 相关指令编号
2	hi_rdata	32	hi 寄存器的值
3	lo_rdata	32	lo 寄存器的值
4	hi_rf_we	1	hi 寄存器是否可写
5	lo_rf_we	1	lo 寄存器是否可写
6	hi_e	1	hi 寄存器是否可读
7	lo_e	1	lo 寄存器是否可读

2.5.3 功能模块说明

算数部分：

对 ID 段传来的 sel_alu_arc 来选择对应操作，并且调用 alu 模块计算 ALUsrc1 和 src2 的结果

```
   assign alu_src1 = sel_alu_src1[1] ? ex_pc :
                     sel_alu_src1[2] ? sa_zero_extend :
rf_rdata1;

   assign alu_src2 = sel_alu_src2[1] ? imm_sign_extend :
                     sel_alu_src2[2] ? 32'd8 :
                     sel_alu_src2[3] ? imm_zero_extend : rf_rdata2;
```

访存部分：

通过获得的各个 inst 值，以及各个指令和写地址的后两位来选择对应的写使能信号。然后根据写使能信号将对应的字节写入 wdata 中最后传到 mycore 模块。

```
assign data_sram_wdata = data_sram_wen == 4'b1000 ?
                       {rf_rdata2[7:0] , 0 , 0 , 0} :
data_sram_wen == 4'b0100 ? {0 , rf_rdata2[7:0] , 0 , 0} :
data_sram_wen == 4'b0010 ? {0 , 0 , rf_rdata2[7:0],0} :
data_sram_wen == 4'b0001 ? {0 , 0 , 0 , rf_rdata2[7:0]} :
data_sram_wen == 4'b1100 ? {rf_rdata2[15:0], rf_rdata2[15:0],
                         0 , 0} :
```

```
data_sram_wen == 4'b0011 ? {0 , 0 ,
                    rf_rdata2[15:0] , rf_rdata2[15:0]} :
                          rf_rdata2;
```

乘除法（hilo 寄存器）部分：

在 EX 段与 hilo 寄存器有关的除了两个简单的读指令（mfhi/mflo）以外主要是与乘除法有关，在这里我们因为乘除法的具体实现是已经实现好了的，因此我们在这里做的更多的是数据通路有关的工作。

乘法的实现主要是通过向 alu 传参后在 alu 内实现，因此传参内容如下，

```
    mul u_mul(
        .clk        (clk           ),
        .resetn     (~rst          ),
        .mul_signed (inst_mult     ),
        .ina        (rf_rdata1     ), // 乘法源操作数 1
        .inb        (rf_rdata2     ), // 乘法源操作数 2
        .result     (mul_result    ) // 乘法结果 64bit
    );
```

除法的实现除了传参以外很重要的是对 stall 的暂停插入，每个除法操作要有 32 位的 stall，因此我们通过 stallreq 来实现

```
reg stallreq_for_div;
assign stallreq_for_ex = stallreq_for_div;
```

ex 段的乘除法运算结束后，我们将其送到下一段，最后由 wb 段传给 ID 后写回寄存器

```
assign hi_rf_wdata = inst_mthi ? rf_rdata1 :
           inst_div | inst_divu ? div_result[63:32] :
           inst_mult | inst_multu ? mul_result[63:32] : 32'b0;
assign lo_rf_wdata = inst_mtlo ? rf_rdata1 :
           inst_div | inst_divu ? div_result[31:0] :
           inst_mult | inst_multu ? mul_result[31:0] : 32'b0;
```

数据相关：

在 EX 段除去各类功能以外我们还需要解决数据相关问题，在不需要 stall 的情况下我们得用 forwarding 技术去除数据相关，我们在 ID 段判断是否需要用到该寄存器的值以解决 RAW 数据相关。

```
    assign ex_wreg=rf_we;
    assign ex_waddr=rf_waddr;
    assign ex_wdata=ex_result;
    assign ex_hi_we=hi_rf_we;
    assign ex_lo_we=lo_rf_we;
```

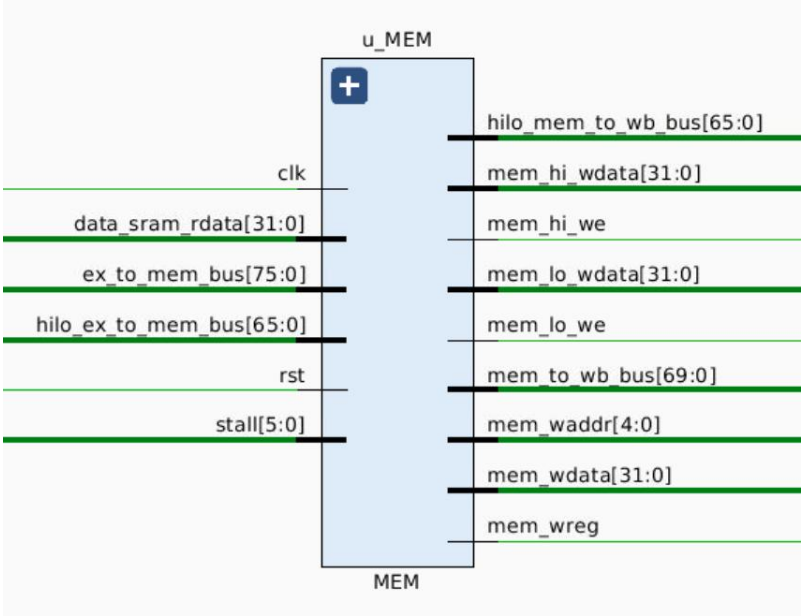
```
assign ex_hi_wdata=hi_rf_wdata;
assign ex_lo_wdata=lo_rf_wdata;
```

2.6 MEM 流水段

2.6.1 功能说明：

MEM 段主要功能是对内存的访问，将 load 指令对应的内存地址所需的字节值从内存中取出出来传到 WB 段。需要定向路径解决数据相关。

2.6.2 信号说明:



序号	名称	I/O	位宽	作用
1	clk	Input	1	时钟信号
2	data_sram_rdata	Input	32	读取出的内存数据
3	ex_to_mem_bus	Input	76	EX->MEM 段总线
4	hilo_ex_to_mem_bus	Input	66	HILO 寄存器 EX->MEM
5	rst	Input	1	复位信号
6	stall	Input	6	延迟信号
7	hilo_mem_to_wb_bus	Output	66	HILO 寄存器 EX->WB
8	mem_hi_wdata	Output	32	MEM 段传回 HI 值
9	mem_hi_we	Output	1	MEM 段是否传回 HI
10	mem_lo_wdata	Output	32	MEM 段传回 LO 值
11	mem_lo_we	Output	1	MEM 段是否传回 LO

12	mem_to_wb_bus	Output	70	MEM->WB 段总线
13	mem_waddr	Output	5	MEM 段传回寄存器号
14	mem_wdata	Output	32	MEM 段传回数据
15	mem_wreg	Output	1	MEM 段局部回路开关

2.6.3 功能模块说明：

MEM 段最主要的操作是读内存。

根据 EX 段总线 ex_to_mem_bus 中从 ID 段传来的 data_ram_wen 判断原操作是哪种操作：

```
wireinst_lw,inst_lb,inst_lh,inst_lbu,inst_lhu;
assigninst_lw
=data_ram_wen==4'b0000?1:0;assigninst_lb=data_ram_wen==4'b1110?
1:0;
assigninst_lh = data_ram_wen==4'b100?1:0;
assigninst_lbu = data_ram_wen==4'b1101?1:0;
assigninst_lhu = data_ram_wen==4'b110?1:0;
```

确认操作种类后，可以通过解析得到读取内存情况：

```
assigndata_sram_wen2=((inst_lb|inst_lbu)&ex_result[1:0]==2'b00)?4
'b0001
:
((inst_lb|inst_lbu)&ex_result[1:0]==2'b01)?4'b0010
:
((inst_lb|inst_lbu)&ex_result[1:0]==2'b10)?4'b0100
:
((inst_lb|inst_lbu)&ex_result[1:0]==2'b11)?4'b1000
:
((inst_lh|inst_lhu)&ex_result[1:0]==2'b00)?4'b0011
:
((inst_lh|inst_lhu)&ex_result[1:0]==2'b10)?4'b1100
:0;
```

其中 ex_result 为自 ex_to_mem_bus 传来的 EX 段 ALU 运算结果，由于读取机制一次读取四个字节，因此应通过其最后两位和指令类型判定取该字节组的某几个字节。（如果指令为 lw 取一整个四个字节，则无需此步骤）作为结果，DATA_SRAM_WEN2 代表要读取四个字节中哪些字节，按自高到低的顺序排列，读取为 1 位，为 0 位不读取。根据预先读取到的 ex_result 处内存片段 data_sram_rdata，按指令要求和 DATA_SRAM_WEN2 取结果：

```
assignmem_result=     inst_lw?data_sram_rdata:
data_sram_wen2==4'b1000?{(inst_lbu?24'b0:
{24{data_sram_rdata[31]}}),data_sram_rdata[31:24]}:
```

```
data_sram_wen2==4'b0100?{(inst_lbu?24'b0:
{24{data_sram_rdata[23]}}),data_sram_rdata[23:16]}:
data_sram_wen2==4'b0010?{(inst_lbu?24'b0:
{24{data_sram_rdata[15]}}),data_sram_rdata[15:8]}:
data_sram_wen2==4'b0001?{(inst_lbu?24'b0:
{24{data_sram_rdata[7]}}),data_sram_rdata[7:0]}:
data_sram_wen2==4'b1100?{(inst_lhu?16'b0:
{16{data_sram_rdata[31]}}),data_sram_rdata[31:16]}:
data_sram_wen2==4'b0011?{(inst_lhu?16'b0:
{16{data_sram_rdata[15]}}),data_sram_rdata[15:0]}:32'b0;
```

其中 lbu 和 lhu 代表读取出的数据片段做无符号扩展，lb 和 lh 代表做有符号扩展。

最后，还要通过 ex_to_mem_bus 提供的导线 sel_rf_res 判断该指令是否涉及读内存运算，如果涉及读内存，便将 mem_result 作为最终的输出结果写入 rf_wdata，然后将 rf_wdata 送入 mem_to_wb_bus 总线输出到 WB 段，再到下一 ID 段写入寄存器。

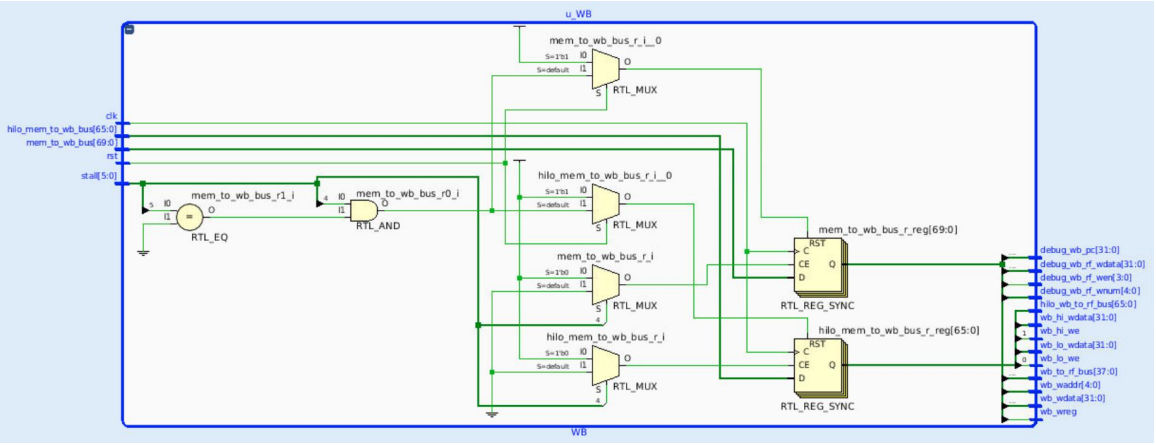
对于 hi 和 lo 两个特殊寄存器，其读写处理已经在 EX 被处理完毕，MEM 段要做的只是传到 WB 段。

2.7 WB 流水段

2.7.1 功能介绍：

WB 段主要功能是写回，将写寄存器的值传到 ID 段，在 ID 段访问 regfile 写入寄存器，同样需要定向路径解决数据相关。

2.7.2 信号说明:



序号	名称	I/O	位宽	作用

1	clk	Input	1	时钟信号
2	rst	Input	1	复位信号
3	stall	Input	6	延迟信号
4	mem_to_wb_bus	Input	70	MEM->WB 段总线
5	hilo_mem_to_wb_bus	Input	66	HILO 寄存器 MEM->WB
6	wb_to_rf_bus	Output	38	WB->寄存器总线
7	hilo_wb_to_bus	Output	66	HILO 寄存器的 WB 线
8	debug_wb_pc	Output	32	调试程序读取 pc 的接口
9	debug_wb_rf_wen	Output	4	调试程序读取是否写寄存器
10	debug_wb_rf_wnum	Output	5	调试程序读取写寄存器号
11	debug_wb_rf_wdata	Output	32	调试程序读取写寄存器值
12	wb_wreg	Output	1	WB 段局部回路开关
13	wb_waddr	Output	5	WB 段传回寄存器号
14	wb_wdata	Output	32	WB 段传回寄存器值
15	wb_hi_we	Output	1	WB 段是否传回 HI
16	wb_lo_we	Output	1	WB 段是否传回 LO
17	wb_hi_wdata	Output	32	WB 段传回 HI 的值
18	wb_lo_wdata	Output	32	WB 段传回 LO 的值

2.7.3 WB 段工作流程：

WB 段有两个内容：

1.向平台的调试程序传送必要信号。

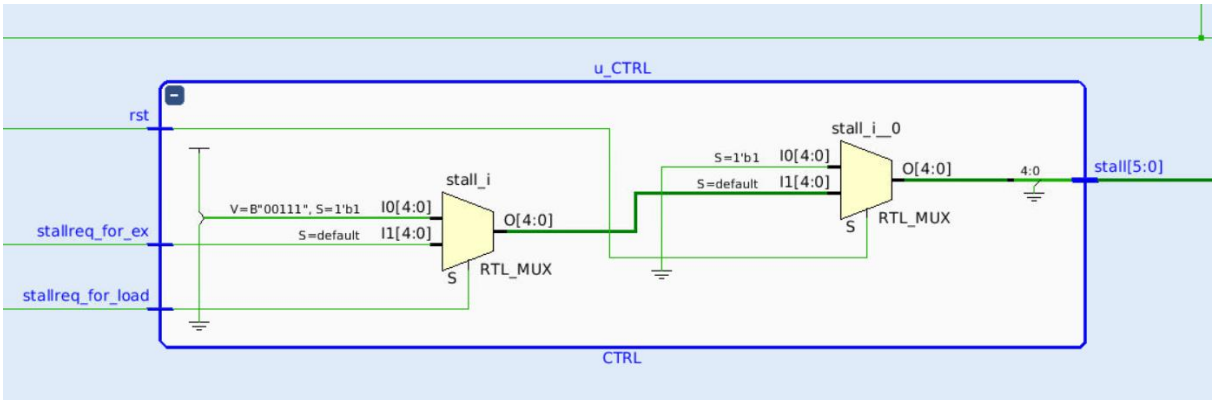
2.向下一个 ID 段的 regfile 组件传入一条 wb_to_rf_bus 和一条 hilo_wb_to_rf_bus，更新 regfile 组件的寄存器数据，起到写入寄存器的作用。

2.8 CRTL 控制段

2.8.1 功能说明

CTRL 段负责的是流水线 stall 暂停的控制，通过接收 ID 段（load 相关指令）和 EX 段（div 指令）发出的暂停请求信号，然后向各段流水发出相 应的暂停信号，完成流水线的暂停控制。

2.8.2 信号说明



序号	名称	位宽	I/O	功能
1	rst	1	Input	复位信号
2	stallreq_for_ex	1	Input	EX 段暂停信号
3	stallreq_for_load	1	Input	ID 段暂停信号
4	stall	6	Output	传出 STALL 信号

2.8.3 功能模块说明

总功能：在收到 stallreq 信号时判断并发出相应暂停信号，6 位二进制数，后五位代表着从 IF 开始每个流水段是否应该 STALL。(这里的六位是按照 stallbus = 6 写的)

判断：

```
always @ (*) begin
       if (rst) begin
           stall = `StallBus'b0;
       end
       else if(stallreq_for_load==1) begin
           stall = 6'b000111;
       end
       else if(stallreq_for_ex==1) begin
                      stall     =  6'b011111 ;
      end
       else begin
           stall = `StallBus'b0;
       end
   end
```

当收到 stallreq_for_load = 1 时，是 load 引起的相关，则在后面插入一个 stall（stall = 000111），暂停一个周期就恢复。

当收到 stallreq_for_ex = 1 时，是 div 运算引起的暂停， 需要暂停整个流水线直到 div 模块计算出结果为止（stall = 011111），共 32 个周期。

2.9 REGFILE 寄存器

2.9.1 功能说明：

regfile 的功能主要是为各类需要用到写寄存器得操作进行写寄存器，而我们主要需要添加的除了数据通路外，主要还是对于 Hilo 寄存器的设置，非 hilo 的数据数据通路的实现方式和 hilo 寄存器也是类似的，因此我们在这里主要介绍 hilo 寄存器设置：

写入：对 req 的赋值，通过 hi/lo_we 写使能线控制是否写入

```
always @ (posedge clk) begin
      if (we && waddr!=5'b0) begin
         reg_array[waddr] <= wdata;
      end
      if (hi_we) begin
         hi_reg <= hi_wdata;
      end
      if (lo_we) begin
         lo_reg <= lo_wdata;
      end
   end
```

读出：通过 ex/mem/we_hi/lo_we 决定是否存在在这些段需要读寄存器的情况（从 MEM、EX 和 WB 三段传来的数组信号调入 regfile 来实现 forwarding），都不是就正常读 hilo 寄存器里的值

```
   assign hi_rdata = (ex_hi_we == 1'b1) ? ex_hi_wdata :
                     (mem_hi_we == 1'b1) ? mem_hi_wdata :
                     (wb_hi_we == 1'b1) ? wb_hi_wdata : hi_reg;

   assign lo_rdata = (ex_lo_we == 1'b1) ? ex_lo_wdata :
                     (mem_lo_we == 1'b1) ? mem_lo_wdata :
                     (wb_lo_we == 1'b1) ? wb_lo_wdata : lo_reg;
```

3. 实验感想

3.1. 魏泽旭实验感想

这次实验可谓是完美验证了一句老话，完事开头难，坦白讲在一开始做完模 60 寄存器时我还以为这东西难度不过如此。但是实际上手才发现其实有很多不同，我自以为的理解有点不知天高地厚了。也是因为一开始东西太繁杂，我们几个人心里其实都打怵，也是因为这个一号点迟迟没能开始，直到十二月 7.8 号吧，我才开始仔细真正的研究波形代码 debug，也在这个过程中我

逐渐的感受到了自己的进步，从最一开始一号点时候每个波形代表的意思不甚清楚要经常问大佬同学，到最后每次 debug 可能很快就能找到故障问题出现在哪，我能切实的感到我对这个东西的理解在逐步加深，虽然在其中也有一些小问题，比如我的实验平台不知道为啥坏了，之后一直是我们三个人用一个人的 cg 在做，不过这也让我们很多时候能够协同作业，还挺好用的。说实话，添加代码，debug，都花费了我大量的时间，我以后很大可能也不会从事硬件方面的工作，但是这次实验让我理解了协作的重要性，以及最重要的，自我摸索的能力，这个东西我们经常会遇到独一无二的 bug 但是也正是在一个个独一无二 bug 的解决中，我能感受到自己的进步。

3.2. 曹宇博实验感想

这次实验通过调试结构的各种问题，有效的锻炼的我应对突发情况排除问题的能力，深化了我对 课程内容的理解。同时，本次实验有效的提高了我的团队协作和沟通能力，考验了我的耐心和细心，让我获得了难以忘怀的回忆。

在这次实验中，我增强了我对 git 和 github 使用方法的掌握能力，同时也强化了我的表达能力和沟 通能力，也强化了我对 verilog 和 vivado 的理解。

3.3. 覃伟境实验感想

由于之前并没有实际上手使用过相关知识，导致本人在此次实验中遇到了很多的问题，归根结底，还是工具使用加上相关知识的不够熟悉。在调试程序 Debug 的过程中，最大的问题是不知道程序到底哪里出了问题，以及该查看哪些参数进行调整。学习这些东西花费了很多的时间和精力，才可以勉强进行操作。在完成项目的过程中，最令我感到痛苦的是不知道该从何下手，虽然之前的完成模 60 有积极思考并认真完成，但模 60 和设计 CPU 之间，从 vivado 这一工具的使用和整个逻辑的实现上，相差了非常大的距离，甚至刚开始做的时候我们连文件导入是否正确都不知道。虽然学长有带着我们熟悉了一下操作，但这对于我们完成项目是远远不够的。

把内心的不快吐露完之后，简单说说整个实验的解决思路吧。首先就是查看相关文档，把一些基本概念弄清楚，然后就是询问一些做过实验的同学该如何进行操作，最后就是和小组的其他成员不断试错，直到解决问题。整个实验的过程中，我都不断的在问自己，这么做的意义是什么？为什么自己要受到这样的折磨？难道是我的能力不行吗？但是当最后和小组的同学一起把 64 号点过了之后，确实感受到了强烈的成就感。

总之，终于把这个折磨人的项目做完了，终于解脱了！

4. 参考资料

1.《自己动手做 CPU》，雷思磊

2.《计算机体系结构》，张晨曦