实验报告成绩:	成绩评定日期:

2024 ~ 2025 学年秋季学期

《计算机系统》必修课

课程实验报告

班级:

组长: 金博

组员: 崔权森

冯育广

报告日期: 2024.12.11

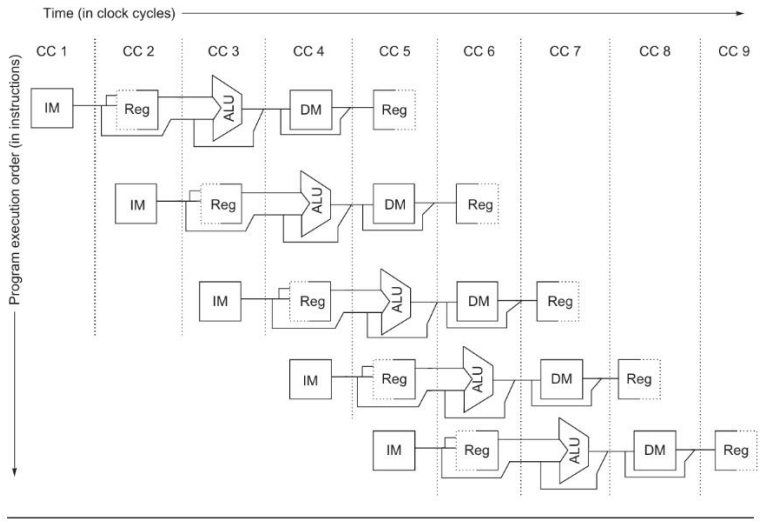
1. 实验概况

1.1 工作量

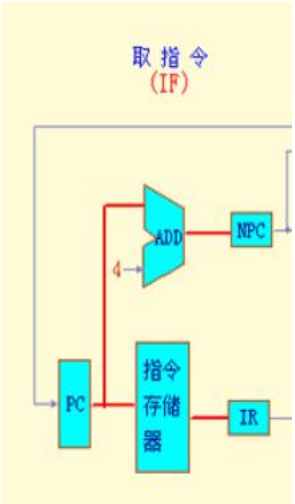
金博　　37%　　崔权森　　33%　　冯育广 30%

1.2 总体设计

处理器的五级流水线架构是一种经典的设计方法，它将处理器的指令执行过程划分为五个阶段：为取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器回写（WB）五级，以提高指令的执行效率和吞吐量。这种五级流水线的设计虽然一个指令的执行需要 5 个时钟周期，但允许多个指令同时处于不同的执行阶段，从而提高了处理器的并行度和效率。



1.2.1IF



取指令周期（IF）

IR ← Mem[PC] Next_PC ← PC＋4

PC 由 MUX 输入

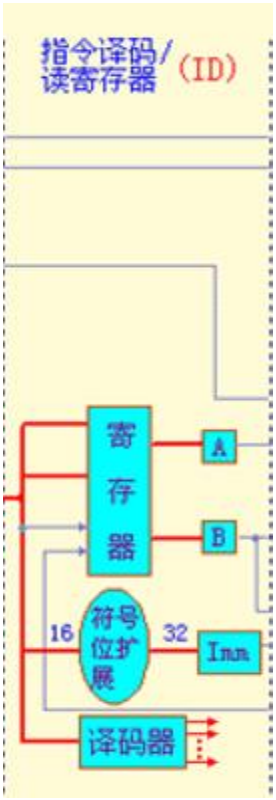
PC 寄存器用于存放指令地址，

IR 寄存器从指令存储器中取出的指令，

NPC 寄存器存储下一个 PC 值(PC+4)，

IF/ID 流水段寄存器存储 IF 的输出（指令和 PC+4）

1.2.2 ID



A ← Regs[IR[6..10]] (Regs[rs])

从 IR 中提取 rs 字段作为索引，获取 Regs[rs]的值到寄存器 A 中

B ← Regs[IR[11..15]] (Regs[rt])

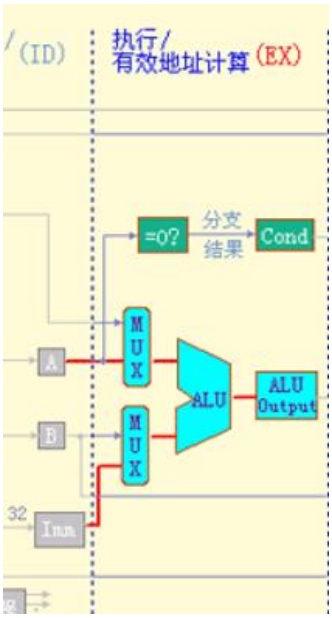
从 IR 中提取 rt 字段作为索引，获取 Regs[rt]的值到寄存器 B 中

Imm ← {IR[16]..31} (Immediate)

从 IR 中提取[16..31]位，获取立即数字段，16 位符号位，通过组合得到 16 位的立即数，存取到 Imm 寄存器。

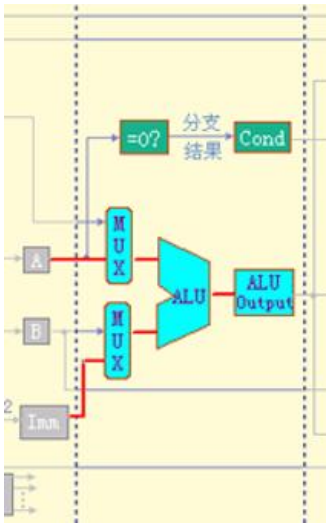
本阶段的主要功能是 ID 段的主要工作为指令的解析，寄存器的访存以及跳转指令的地址计算。将指令进行译码生成控制信号，并从寄存器中取出相应的操作数。是从上一个流水段寄存中取出指令，并送入译码控制器进行译码，译码完成厚将译码结果存入下一个流水段寄存器，并根据译码结果将立即数扩展，扩展方式共有 4 中，分别是逻辑扩展，算数扩展，为 lui 指令进行的扩展，和为分支指令进行的扩展。

1.2.3 EX



存储器访问（load 和 store）

ALUOutput ← A＋Imm

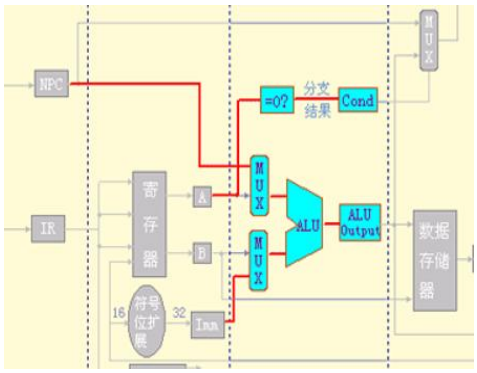


寄存器－寄存器 ALU 操作

ALUOutput ← A op B

寄存器－立即值 ALU 操作

ALUOutput ← A op Imm



分支操作

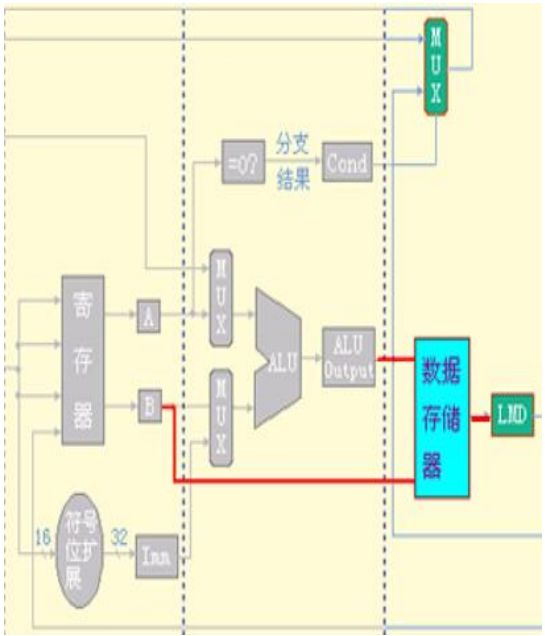
ALUOutput ← NPC＋Imm //计算偏移地址

Cond ← (A op 0) //判断分支是否成功，失败则结束

根据指令的编码进行算数或者逻辑运算或者计算条件分支指令的跳转目标地址。此外 LW、SW 指令所用的 RAM 访问地址也是在本周期上实现。

数据运算是从上一流水段寄存器中取出操作数和控制信号，根据控制信号控制数据通路进行数据的运算，并将运算的结果存入下一个流水段。但是其取出的操作数也是进行了转发选择的数据，转发同样还是采用的按字节转发的模式。

而抵制运算则是为了跳转和分支指令准备的，如果跳转地址会被送回第一部分也即取值模块的 PC 源之一进行选择。跳转指令也会再下一个流水段送回，因为在下一个流水段才能知道是否需要跳转



1.2.4 MEM

在该周期处理的DLX指令只有Load、Store和分支指令存储器访问(load和store)
其他类型指令均不做

LMD ← Mem[ALUOutput] 或

Mem [ALUOutput] ← B

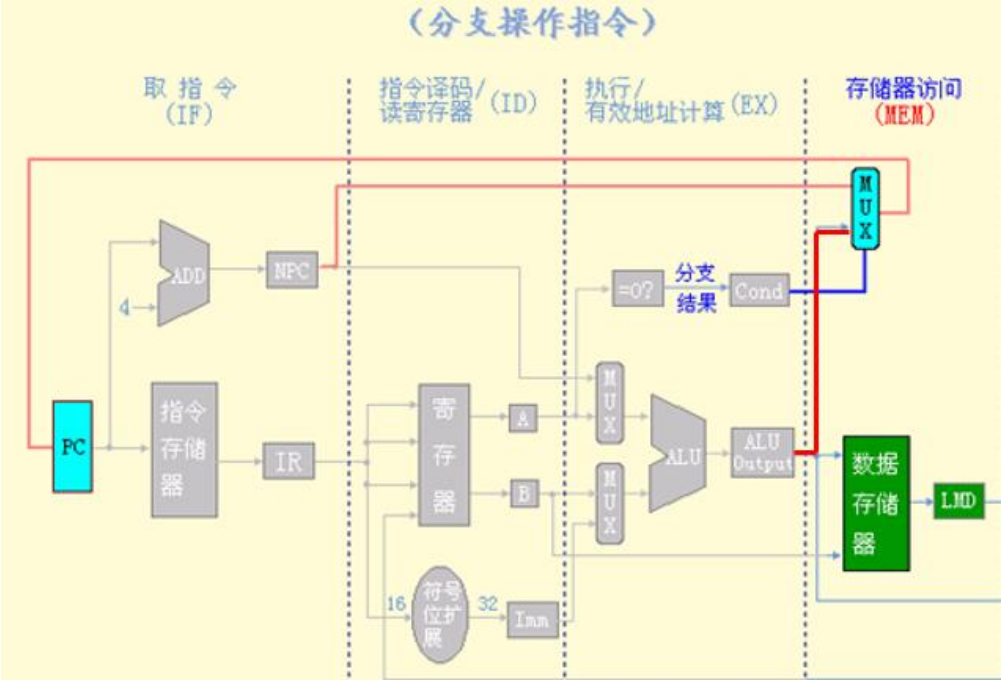
分支操作

if (cond) PC ← ALUOutput

else PC ← NPC

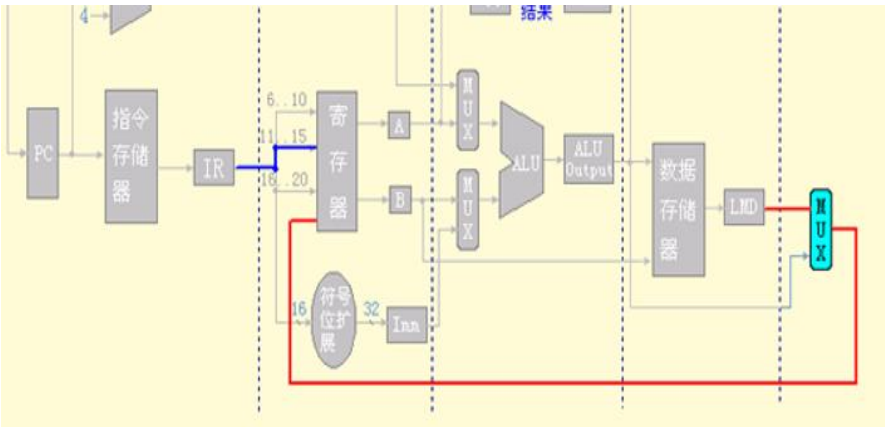
成功则把计算好的地址放入PC

否则不做



只有在执行LW、SW指令时才对存储器进行读写，对其他指令只起到一个周期的作用。本阶段进行的操作是内存相关的操作，在内存的读写口之前都有一个移位器，只是为LW和SW等类型的指令准备的，还有一个condition的检查单元，该单元是用于判断跳转是否有效和是否真的需要写回寄存器的

1.2.5 WB



不同指令在该周期完成的工作也不一样

寄存器—寄存器型 ALU 指令

Regs[IR16 ..20] (rd)← ALUOutput

寄存器—立即值型 ALU 指令

Regs[IR11 ..15] (rt)← ALUOutput

Load 指令

Regs[IR11 ..15] (rt)← LMD

该段把指令执行的结果写回到寄存器文件中，即 ALU 运算指令和 load 指令在这个周期把结果数据写入通用寄存器组。ALU 运算指令的结果数据来自 ALU,而 load 指令的结果数据来自存储器。 整个流水线 CPU 的时钟关系，由于每个流水段寄存器都是设计在下降沿进行写入 的，而寄存器的读取设计在上升沿，为了解决写回寄存器的时候可能同时对同一地址 进行读写引起的结构冲突，所以在设计的时候采用了读写分不同边沿的，那么这里就 有可能出现数据的延迟，也就是想要的数据还没写回即要读取，因此这里会使用到一 种转发关系。

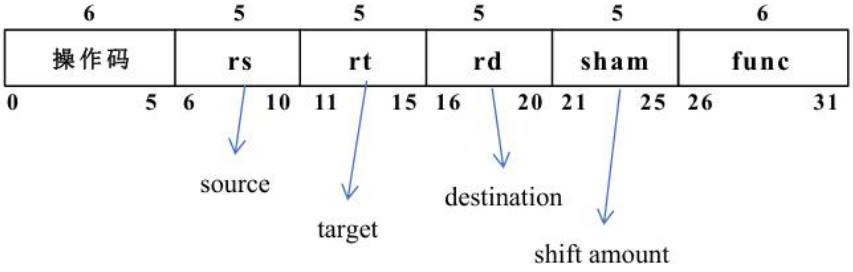
分支指令需要 4 个时钟周期（移到 ID 段，只需 2 个周期

store 指令需要 4 个时钟周期 其它指令需要 5 个时钟周期

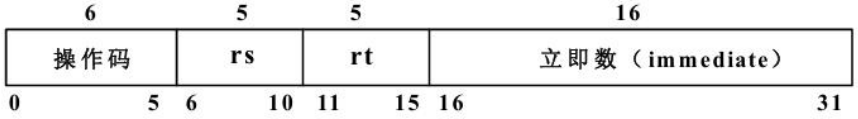
1.3 MIPS

本实验选用 MIPS-32

R 类指令：



I 类指令：



J 类指令：

1.4 指令完成度

添加完成了如下指令： 算术运算指令、逻辑运算指令、移位指令、分支跳转指令、数据移动指令、访存指令， 共计 49 条指令。 按照写实验的先后循序这些指令顺序如下： inst_ori, inst_lui, inst_addiu, inst_beq, inst_subu, inst_jr, inst_jal, inst_addu, inst_bne, inst_sll, inst_or, inst_lw, inst_sw, inst_xor , inst_sltu, inst_slt, inst_slti, inst_sltiu, inst_j, inst_add, inst_addi , inst_sub, inst_and , inst_andi, inst_nor, inst_xori, inst_sllv, inst_sra, inst_bgez, inst_bltz, inst_bgtz, inst_blez, inst_bgezal, inst_bltzal, inst_jalr, inst_mflo, inst_mfhi, inst_mthi, inst_mtlo, inst_div, inst_divi, inst_mult, inst_multu, inst_lb, inst_lbu, inst_lh, inst_lhu, inst_sb, inst_sh 成功通过第 64 个点。

2. 流水段说明

2.1IF

功能：从指令存储器中读取指令，并根据分支信号决定下一条指令的地址。

输入：

clk：时钟信号。

rst：复位信号。

Stall [5:0]：流水线暂停信号。从 CTRL.v 接收，[0]=1’b1 暂停

br_bus [32:0]：分支信号总线，包含 br_e 和 br_addr。1 跳转，跳转时 br_addr 为新的 PC 值

输出：

if_to_id_bus [32:0]：是将当前得到的 pc 值发送给 ID 段。

inst_sram_en：是进行读指令存储器的使能信号。

inst_sram_wen [3:0]：指令存储器写使能信号（0，表示只读）。

inst_sram_addr [31:0]：是将存入了当前的 pc 值，并将其发送给指令存储器，指 令存储器得到了 pc 值后，将改 pc 值对应下的指令 inst 值发送给 ID 段中，进行译码。

inst_sram_wdata [31:0]：指令存储器写数据（0，表示只读）。

其他：

pc_reg：当前指令的 PC 值。

next_pc：下一条指令的 PC 值，根据分支信号决定。

ce_reg：指令存储器使能信号。

2.2 ID 段

功能：ID 段对指令进行译码，读取寄存器文件，生成控制信号，进行符号扩展或无符号扩展，检测数据冒险，并处理跳转指令。当指令为跳转指令时，ID 段计算跳转目标地址，并通过 `br_bus` 将跳转信息传回 IF 段。

输入端口：

clk：时钟信号。

rst：复位信号。

stall：流水线暂停信号。`stall[5:0]` 由控制逻辑产生，用于暂停流水线中的各个阶段。

always @ (posedge clk) begin //如果 ID 段需要暂停，则将当前的 inst 值赋值给 inst_stall 临时寄存器中

```
        inst_stall_en<=1'b0;
        inst_stall <=32'b0;
        if(stall[1] == 1'b1 & ready_ex_to_id ==1'b0)begin
        inst_stall <= inst;//
        inst_stall_en<=1'b1;
        end
    end
    assign inst_stall1 = inst_stall;//将上一个时钟周期 inst_stal 的值
    assign inst_stall_en1 = inst_stall_en ;// 赋给当前的 inst 值
    assign inst = inst_stall_en1 ? inst_stall1 :inst_sram_rdata;
```

ex_to_id_bus[37:0]、mem_to_id_bus[37:0]：EX、MEM 段传递给 ID 段的数据。

ex_to_id_2[65:0],mem_to_id_2[65:0],wb_to_id_2[65:0]

if_to_id_bus[32:0]：IF 段传递过来的当前指令的 PC 值。

inst_sram_rdata[31:0]：从指令存储器读取的数据。

inst_is_load:产生停顿请求:inst_is_load 被用于生成停顿请求信号 stallreq_for_id。具体而言，当 inst_is_load 为 1 且当前指令涉及到的源寄存器（rs 或 rt）与 EX 到 ID 阶段的数据总线中的热寄存器冲突时，将请求停

顿。这样可以防止数据冒险，从而确保数据的正确性。其判断了如果当前指令是一个加载指令，并且其操作的寄存器正在由执行阶段的指令使用，那么就需要请求停顿，避免数据不一致的问题。

wb_to_rf_bus[37:0]：WB 段传递给寄存器文件的写回数据。

wb_to_id_wf[65:0] 用于写入高寄存器（HI）和低寄存器（LO）的控制信号

ready_ex_to_id：EX 段传递给 ID 段的信号，表示乘除法等操作是否完成。

输出端口：

br_bus[32:0]：分支信号总线，ID 段生成的分支信号。

id_to_ex_bus[230:0]：ID 段传递给 EX 段的总线，包含译码后的控制信号和操作数。

stallreq_for_id：请求暂停 ID 段的信号，通常用于处理数据冒险。

其他：

inst：当前指令。

id_pc：当前指令的 PC 值，用于标识该指令的位置。

rdata1, rdata2：从寄存器文件读取的两个操作数，通常来自指令中的 rs 和 rt 字段。

alu_op：ALU 操作码，用于指示 ALU 执行的操作类型。

sel_alu_src1, sel_alu_src2：ALU 操作数选择信号，决定 ALU 输入的操作数来源。

br_bus：分支跳转指令的跳转目标地址及跳转使能信号，传递给 IF 段。

stallreg_for_id：请求暂停 ID 段的信号，通常用于数据冒险。

数据冒险检测：

如果 EX 段的 LW 指令与当前指令的寄存器读取发生冲突，ID 段会生成暂停信号 `stallreq_for_id`，请求暂停流水线以解决数据冒险。

2.3 regfile.v

在 ID 段，可以对 regfile.v 文件进行操作，regfile.v 文件中包括了寄存器 reg_array[31:0]和存乘除法结果的 hilo 寄存器。

主要的输入信号：

r_hi_we,

r_lo_we,

raddr1[4:0],

raddr2[4:0]

主要的输出信号：

hi_o[31:0],

lo_o[31:0],

rdata1[31:0],

radta2[31:0]

其中 r_hi_we 和 r_lo_we 是读取 hilo 寄存器的信号，其从 hilo 寄存器中读出的结果赋值给 hi_o[31:0]和 lo_o[31:0]返回给 ID 段。

其中 raddr1[4:0],raddr2[4:0]是要读取 reg_array[31:0]寄存器中的地址，其从 reg_array[31:0]寄存器读出的结果赋值给 rdata1[31:0]和 radta2[31:0]返回给 ID 段。

2.4 EX

功能：执行算术逻辑运算，处理乘除法操作，生成访存地址和写回数据。

输入端口：

clk：时钟信号。

rst：复位信号。

stall[5:0]：流水线暂停信号。如果 stall[3]==1’b1，则代表要暂停 EX 段，当 EX 段的指令为乘法或除法时，因为乘除法需要 32 个时钟周期的计算时间，所以需要 IF、ID、EX 段进行暂停操作，直到乘除法指令结束后流水 线才正常运行。

id_to_ex_bus[230:0] 是 ID 段要发送给 EX 的值。

```
assign {
    ex_pc,          // 158:127
    inst,           // 126:95
    alu_op,         // 94:83进行alu_op操作的信号
    sel_alu_src1,   // 82:80alu操作数1的目标值选择sel_alu_src1
    sel_alu_src2,   // 79:76alu操作数2的目标值选择sel_slu_src2
    data_ram_en,    // 75对存储器进行访存操作的使能信号
    data_ram_wen,   // 74:71对存储器进行操作
    rf_we,          // 70对寄存器进行写操作的使能信号
    rf_waddr,       // 69:65寄存器进行写操作的寄存器的地址
    sel_rf_res,     // 64
    rf_rdata1,         // 63:32      //rs
    rf_rdata2,          // 31:0      //rt
    lo_hi_r,                        //read
    lo_hi_w,                        //write
    lo_o,                           //lo
    hi_o,                            //hi
    data_ram_read   //对寄存器进行操作
} = id_to_ex_bus_r;
```

输出端口：

data_sram_addr[31:0]：将 EX 段中算出的结果传给存储器进行寻址，并将寻址得到的值通过 data_sram_rdata 传递到 MEM 段中。data_sram_en：对存储器的访存使能信号。

data_sram_wdata[31:0]：数据存储器写数据。

data_sram_wen[3:0]：数据存储器写使能信号。通过传递当前指令对存储器的操作指令，在 data_sram_wdata[31:0]中控制不同类型的将要写入存储器的值。

ex_to_id_bus[37:0]：是跟数据相关有关的指令，当前指令需要取前面还未存入寄存器的值的时候，EX 段提前发给 ID 段，再由 ID 段发送给 regfile.v 文件中，进行赋给 rs 和 rt 所需要的寄存器的值。

ex_to_id2[65:0]：是 EX 段要发送给 ID 段中的 regfile.v，用于解决下一条指令 要用到上一条指令存入 hilo 寄存器值的问题。

ex_to_mem_bus[79:0]：EX 段传递给 MEM 段的总线，包含执行结果和访存控制信号。

ex_to_mem1[65:0]：是 EX 段要发送给 MEM 的值，其包括了写 hi 和 lo 寄存器的使能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo 寄存器的值，如果不写，则此处为 0，且使能信号为 0。

inst_is_load：是 EX 段发送到 ID 段的信号，用来判断 EX 的当前指令是否是 LW 指令，如果是，则该值为 1，如果不是，则该值为 0。并且与 rs 寄存器和 rt 寄存器 中的地址进行判断，如果 EX 段的 lw 要写入的寄存器的地址与当前 ID

段的指令要读取的寄存器的地址有相同的，则 stallreg_for_id 为 1’b1，并将此值赋值给 CTRL.v 中，在此文件中发出暂停 ID 段和 IF 的暂停信号。

ready_ex_to_id：是 EX 段发送到 ID 段的一个信号，用来接收到 EX 段中乘除法操作是否完成，如果没有完成，则该信号值为 0，如果完成该信号的值为 1，如果在 ex 段进行的指令是乘除法，因为他们要进行 32 个时钟周期，所以要将后面的流水段进行暂停，所以 ID 段需要一直保存上个时钟周期的 inst 值。

stallreq_for_ex：是从 EX 段发送给 CTRL.v 的一个请求暂停指令，这个操作是由于乘除法器需要 32 个时钟周期计算而导致的，需要让 CTRL.v 发送到 IF、ID、EX 段进行暂停操作，直到乘除法指令结束后流水线才正常运行。

其他：

alu_result：ALU 运算结果。

ex_result：执行结果，可能是 ALU 结果或乘除法结果。

mul_result, div_result：乘除法结果。

2.5 MEM

功能：如果是 load/store 指令，那么在此阶段会访问数据存储器，反之，只是将执行 阶段的结果向下传递到回写阶段。如果存储器传回来了数值，则需要进行通过相关的 指令如 lb、lbu、lh、lhu、sb、sh 来判断进行取值操作，再进一步将存储器得到的 值存入到寄存器中。同时，在此阶段还要判断是否有异常需要处理，如果有，那么会 清除流水线，然后转移到异常处理例程入口地址处继续执行。 MEM 段最后会将各类寄存器的读写使能信号、地址和写入数据合并为 MEM_TO_WB 总线，并传入 WB 段。

输入端口：

clk：时钟信号。

rst：复位信号。

stall[5:0]：流水线暂停信号。

ex_to_mem1[65:0]：是 MEM 接收到从 EX 段发送过来的值，其包括了写 hi 和 lo 寄存器的使能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo 寄存器的值，如果不写，则此处为 0，且使能信号为 0。

ex_to_mem_bus[79:0]：EX 段传递过来的执行结果和访存控制信号。

data_sram_rdata[31:0]：是从存储器中在 EX 读取到的数值，在 MEM 接收得到。

输出端口：

mem_to_wb_bus[69:0]：MEM 段传递给 WB 段的总线，包含访存结果和写回数据。

```
assign {
    mem_pc,          // 75:44
    data_ram_en,     // 43 对存储器进行访存操作的使能信号
    data_ram_wen,    // 42:39 对存储器进行操作
    sel_rf_res,      // 38
    rf_we,           // 37 写寄存器使能
    rf_waddr,        // 36:32 写操作寄存器地址
    ex_result,        // 31:0 ex计算的结果
    data_ram_read     // 对寄存器进行操作
} =  ex_to_mem_bus_r;
```

mem_to_id_bus[37:0]：]是跟数据相关有关的指令，当当前指令需要取前面还未存入寄存器的值的时候，由 MEM 段提前发给 ID 段，再由 ID 段发送给 regfile.v 文件中， 进行赋给 rs 和 rt 所需要的寄存器的值

mem_to_id_2[65:0]：是 mem 段要发送给 ID 段中的 regfile.v，用于解决下一条指 令要用到上一条指令存入 hilo 寄存器值的问题。

mem_to_wb1[65:0]：是 MEM 段要发送给 WB 的值，其包括了写 hi 和 lo 寄存器的使 能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo 寄存器的 值，如果不写，则此处为 0，且使能信号为 0。

其他：

在 MEM 段中，会进行判断一下，最终写入寄存器中的值是从 EX 段传过来的 ex_result[31:0]还是从存储器中传下来的 data_sram_rdata[31:0],判断后再传给 rf_wdata。

```
assign mem_result = data_sram_rdata;

assign rf_wdata =  (data_ram_read==4'b1111 && data_ram_en==1'b1) ? mem_result :…
```

2.6 WB

功能：将运算结果保存到目标寄存器。

输入端口：

clk

rst

stall[5:0]

mem_to_wb_bus[69:0]

```
assign {
    wb_pc,
    rf_we,
    rf_waddr,
    rf_wdata
} = mem_to_wb_bus_r;
```

mem_to_wb1[65:0]

```
assign
{
    w_hi_we,
    w_lo_we,
    hi_i,
    lo_i
} = mem_to_wb1_r;
```

输出端口：

wb_to_id_bus[37:0] 是跟数据相关有关的指令，当当前指令需要取前面还未存入 寄存器的值的时候，由 WB 段提前发给 ID 段，再由 ID 段发送给 regfile.v 文件中， 进行赋给 rs 和 rt 所需要的寄存器的值

wb_to_id_wf[65:0] 是 WB 段要发送给 regfile.v 的值，其包括了写 hi 和 lo 寄 存器的使能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo 寄存器的值，如果不写，则此处为 0，且使能信号为 0。

wb_to_id_2[65:0] 是 WB 段要发送给 ID 段中的 regfile.v，用于解决下一条指令 要用到上一条指令存入 hilo 寄存器值的问题。

wb_to_rf_bus[37:0] 是 wb 要写回 reg_array[31:0]的值，其中包括了对寄存器 reg_array[31:0]进行写操作的使能信号 rf_we,对寄存器 reg_array[31:0]进行写操 作的寄存器的地址 rf_waddr，EX 段中算出的结 ex_result[31:0]。

3. 实验心得

3.1 金博

开始做计算机系统实验的这段时间，正好是各种实验课和课设一起来，的确是非常急迫的任务。刚开始入门的时候，真的是看着代码两眼一抹黑不知道怎么调整。后面再实验 ppt 的帮助下和同学的帮助下慢慢的通过一个个 point，尝试了使用波形图、test.s 文件进行 debug，通过比较 PC 值、wnum 和 wdata 来确认是哪块出现了问题。比如在添加连线、指令以及停止操作后，发现一直卡在 point38 过不去，在这里折腾了好久才发现是 ID 段的 BLEZ 指令写错了。通过这次实验，我对 CPU 有了更多的理解，同时也感受到细节决定成败。

3.2 崔权森

计算机系统这门课时间紧任务重，实验入门开始都是两眼一抹黑。Point1 都不好过刚开始对于宏的定义都不了解而且我们都是第一次接触真的很难，而且我们一点一点从补全连线到添加指令从添加 data_ram_read 到高低位寄存器的修改，我们学会了 debug 和波形图还有通过指令对比看我们那块出问题了，每一步都是在学习，每一步都是在理解，因为这门课让我懂了更多，学到了更多，纸上得来终觉浅，绝知此事要躬行。

3.3 冯育广

计算机系统课程的这门实验，首先就让我感觉入门很难，一切的一切都是崭新的接触，再加上时间紧任务重，很崩溃。但我们小组还是一边学习流水线，一边恶补相关知识，从一开始的接线到过完 8 个点，虽然遇到了无数的困难，但在同学的帮助下，我慢慢学会了通过波形图来 debug，并通过对比 mycpu 和 reference 值的不同之处来推断是哪里的问题，暂停亦或者指令。同时我也理解了很多不同变量的作用。后面的编写也更加轻车熟路。通过此次实验，让我们对 CPU 有了更深的理解，而不只是纸上谈兵。