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 1901

组长：严洁

组员：王馨霄、魏钞迪

报告日期：2021.12.18

目　录

1. 成员分工

1.1 工作占比顺序

严洁、王馨霄、魏钞迪

1.2 成员工作量

严洁：通过 Point 1-43，完成数据搬移指令，实现了复用了主要逻辑可综合的乘除法器。
王馨霄：通过 Point 1-64。
魏钞迪：通过 Point 1-43。

2. 总体设计

2.1 总体功能模块

为实现 CPU 预期功能，首先需要对流水线系统结构进行总体功能设计。各个阶段设计的主要工作如下。
(1)取指(IF 段)：取出指令寄存器中的指令，PC 值递增，准备取下一条指令。
(2)译码(ID 段)：对指令进行译码，依据译码结果，从 32 个通用寄存器中取出源操作数，根据指令的不同要求，利用两个复用器，确定参与运算的操作数，最终确定的两个操作数送入下一级流水段。
(3)执行(EX 段)：依据译码阶段送入的源操作数、操作码，进行运算，并将运算结果传递到访存阶段。
(4)访存(MEM 段)：根据译码结果和 EX 段的运算结果，对需要访存的指令，进行存储器内容操作，包括读、写操作。
(5)回写(WB 段)：将运算结果保存到目的寄存器。
图 2-1 为设计而成的流水系统结构图，图中显示了各个模块接口、连接关系。每个模块上方是模块名。需要注意的是，图中 ctrl 模块的主要工作为，提供暂停，保证流水线正常运行。

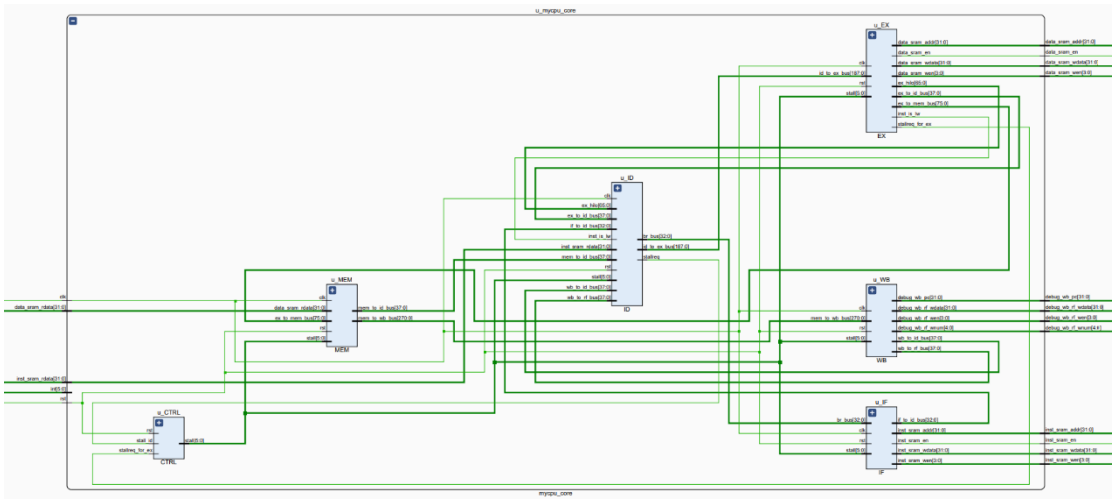

图 2.1-1 五级流水段系统结构图

2.2 指令

处理器实现的指令包括算术运算指令、逻辑运算指令、条移位指令、分支跳转指令、数据移动指令和访存指令，下表为其具体实现的各类指令。

指令类别	指令
算术运算指令	ADD,ADDI,ADDU,ADDIU,SUB,SUBU,SLT,SLTI,SLTU,SLTIU,DIV,DIVU,MULT,MULTU
逻辑运算指令	AND,ANDI,LUI,NOR,OR,ORI,XOR,XORI
移位指令	SLL,SLLV,SRA,SRAV,SRL,SRLV
分支跳转指令	BEQ,BNE,BGEZ,BGTZ,BLEZ,BLTZ,BLTZAL,BGEZAL,J,JAL,JR,JALR
数据移动指令	MFHI,MFLO,MTHI,MTLO
访存指令	LB,LBU,LH,LHU,LW,SB,SH,SW

表 2.2-1 处理器实现的各类指令

2.3 程序运行环境及使用工具

2.3.1 程序运行环境

(1)计算机系统：Windows10 操作系统

(2)CPU: Core i7

2.3.2 使用工具

(1)Vivado 2019.2

(2) Visual Studio Code

3. 详细设计

3.1 IF 段

3.1.1 整体功能说明

取出指令寄存器中的指令，PC 值递增，准备取下一条指令。

3.1.2 端口介绍

(1)输入端口：

端口名	位宽	作用
clk	1	时钟信号
rst	1	复位信号
stall	StallBus	暂停
br_bus	BR_WD	跳转总线

表 3.1.2-1 IF 段输入端口表

(2)输出端口：

端口名	位宽	作用
if_to_id_bus	IF_TO_ID_WD	IF 段到 ID 段总线
inst_sram_en	1	指令存储器使能信号
inst_sram_wen	4	指令存储器写使能信号
br_bus	BR_WD	跳转总线

表 3.1.2-2 IF 段输出端口表

3.1.3 功能模块说明

IF 段主要功能如下：

(1)复位处理

在复位时,寄存器 pc_reg 赋值为’bfbf_fffc’，寄存器 ce_reg 赋值为 0；在复位信号为 0 且暂停当前段不暂停时,寄存器 pc_reg 与线路 next_pc 连接，寄存器 ce_reg 赋值为 1。

(2)跳转处理

当跳转使能信号为 1，即需要跳转时，线路 next_pc 与线路 br_addr 连接；否则线路 next_pc 与寄存器 pc_reg+4 连接。

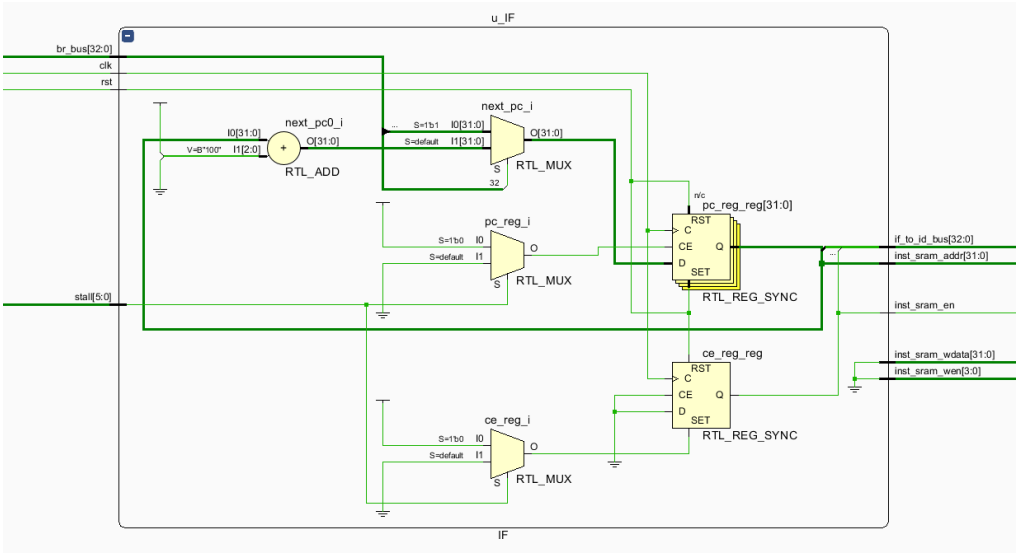

图 3.1.3-1 IF 段结构模型图

3.2 ID 段

3.2.1 整体功能说明

(1)对指令进行译码

(2)根据指令类型取得源操作数并将其传给 EX 段

(3)对于跳转指令，计算其对应跳转地址

3.2.2 端口介绍

(1)输入端口

端口名	位宽	作用
clk	1	时钟信号
rst	1	复位信号
stall	`StallBus	暂停状态
ex_to_id_bus	38	EX 段到 ID 段的线束
mem_to_id_bus	38	MEM 段到 ID 段的线束
wb_to_id_bus	38	WB 段到 ID 段传递的线束
inst_is_load	1	访存指令判定信号
if_to_id_bus	`IF_TO_ID_WD	IF 段到 ID 段的线束
inst_sram_rdata	32	指令内存读取接口
wb_to_rf_bus	`WB_TO_RF_WD	写回寄存器的线束
ex_hilo	66	EX 段到 ID 段 hi、lo 寄存器相关线束

表 3.2.2-1 ID 段输入端口表

(2)输出端口

端口名	位宽	作用
stallreq	1	暂停请求信号
id_to_ex_bus	`ID_TO_EX_WD	ID 段到 EX 段的线束
br_bus	BR_WD	跳转指令线束

表 3.2.2-2 ID 段输出端口表

3.2.3 信号介绍

(1)stallreq(暂停请求信号)：用于发现定向方法解决不了的数据相关，具体指一条访存指令后接一条源寄存器是访存指令目的寄存器的指令。这一暂停的发射条件为 EX 段输入的 inst_is_load 为 1，且 EX 段返回的目的寄存器和当前指令的任何一个源寄存器相同，该信号发射后将通过改变 stall 数组的状态在 MEM 段插入暂停气泡。

(2) wb_rf_we (寄存器写使能信号)：由 WB 段输入，当 wb_rf_we 为 1 时，按 WB 段返回的寄存器地址写入相应数据。

(3)ex_we(EX 段写使能信号)、mem_we(MEM 段写使能信号)、wb_we(WB 段写使能信号)：分别由 EX、MEM、WB 段输入，用于判断是否使用定向的方法解决通用寄存器数据相关。以 EX 段定向为例，当 ex_we 为 1，且 EX 段返回的目的寄存器和当前指令的任何一个源寄存器相同，就说明此时通用寄存器出现了数据相关问题，使用 EX 段返回的数据代替原本从寄存器中读取的数据。MEM 段、WB 段同理，且优先级上 EX > MEM > WB。

(4)hi_we,lo_we(hi、lo 寄存器写入使能信号)：由 EX 段输入，用于判断是否向 hi、lo 寄存器中写入数据，当 hi_we(lo_we)为 1 时，按 EX 段返回的寄存器地址写入相应数据。此外，它们还用于判断是否使用定向的方法解决 hi、lo 寄存器数据相关。以 hi 寄存器为例，当 hi_we 为 1，且当前指令为 mfhi，此时 hi 寄存器出现了数据相关问题，使用 EX 段返回的数据代替原本从 hi 寄存器中读取的数据，lo 寄存器同理。

3.1.4 功能模块说明

1.实例化对象说明

(1) u_regfile：用于读写通用寄存器，通用寄存器为 32 个 32 位的寄存器，具体定义形式为

reg [31:0] reg_array [31:0];

有一个写端口和两个读端口，其原理图如下所示

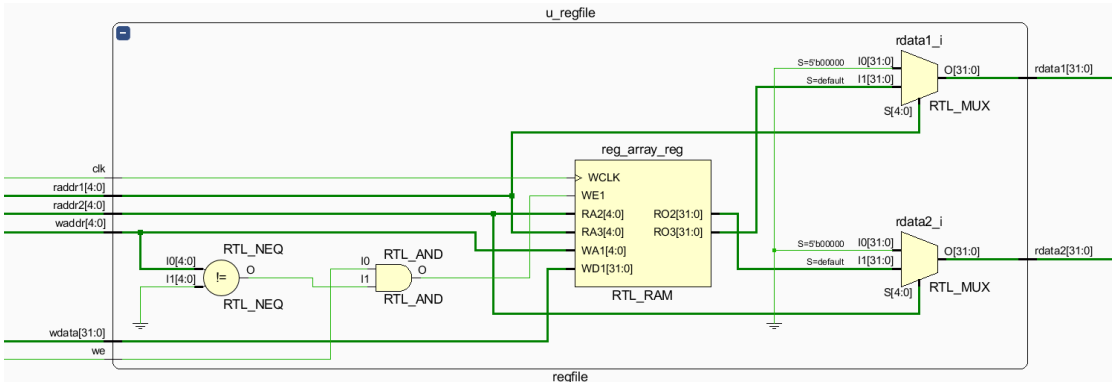

图 3.1.4-1 通用寄存器原理图

(2) u_lo_regfile：用于读写 hi、lo 寄存器，hi、lo 寄存器均为 32 位的寄存器，具体定义形式为

```
reg [31:0] lo_reg;
reg [31:0] hi_reg;
```

有一个写端口和一个读端口，多路复用器的信号和寄存器的 CE 信号分别为其读使能和写使能信号，其原理图如下所示

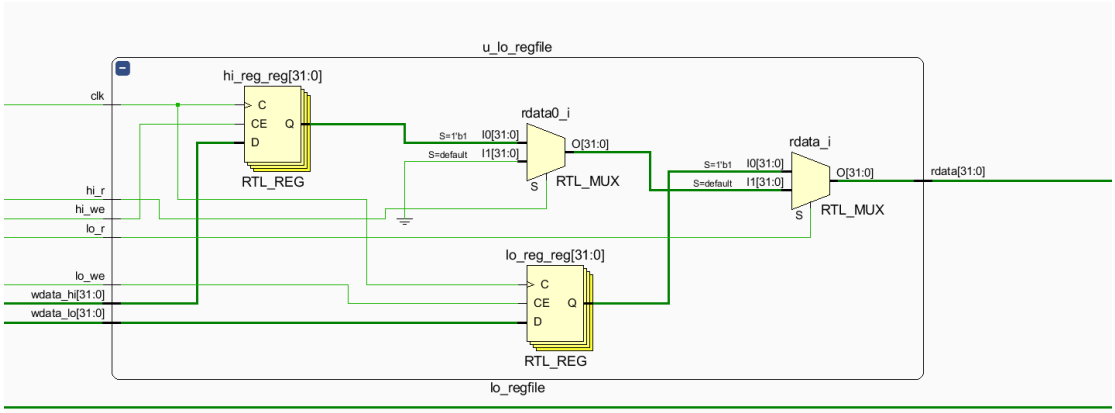

图 3.1.4-2 hi、lo 寄存器原理图

(3)decoder：在本段这类对象有 2 类 4 个—u0_decoder_6_64、u1_decoder_6_64、u0_decoder_5_32、u1_decoder_5_32，分别用于将 opcode(6 位)、func(6 位)、rs(5 位)、rt(5 位)转换成各自的独热形式，这一形式将用于后续的译码。

2.工作流程说明

ID 段主要分为读取指令、指令译码、写寄存器、获取 EX 段输入线束、跳转模块五个模块，总体工作流程如下图所示

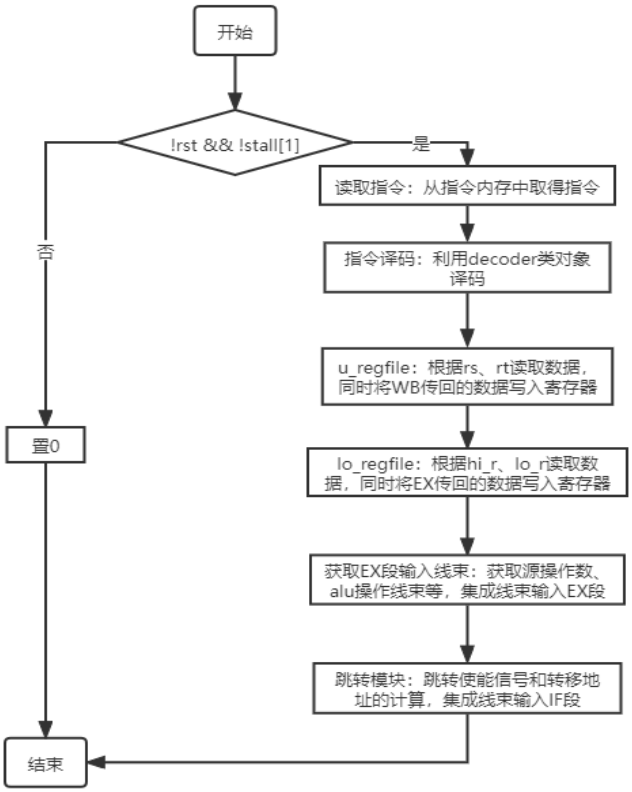

图 3.1.4-3 ID 段工作流程示意图

(1)读取指令

在正常不发生暂停的情况下，直接从指令内存中读出指令即可；但因为处理器直接用了存储器当指令的级间寄存器，暂停的时候会有一条指令丢失，因此在这种情况下，需要加一个类似于寄存器的器件将这条指令暂时保存下来，这部分代码如下所示

```
reg flag;
    always @ (posedge clk) begin
    if (stall[2]==`Stop) begin
        flag <= 1'b1;
    end
    else begin
        flag <= 1'b0;
    end
end
assign inst = flag?inst:inst_sram_rdata;
```

(2)指令译码

利用 u_decoder 将 opcode(6 位)、func(6 位)、rs(5 位)、rt(5 位)转换成各自的独热形式，由它们分别判断当前指令是哪一种指令，从而确定如何取得操

作数、是否需要写寄存器及写回地址等信息。

(3)读写寄存器

通用寄存器：利用 u_regfile 根据 rs、rt 读取数据，同时将 WB 传回的数据写入寄存器；hi、lo 寄存器：利用 lo_regfile 根据 hi_r、lo_r 读取数据，同时将 EX 传回的数据写入寄存器。

(4)获取 EX 段输入线束

由 ID 段输入 EX 段的线束中，在 ID 中得出的量有 alu 输入线束、源操作数选择方式线束、2 个源操作数、通用寄存器写使能、通用寄存器写入地址

alu 输入线束：op_add, op_sub, op_slt, op_sltu, op_and, op_nor, op_or, op_xor, op_sll, op_srl, op_sra, op_lui;分别代表 alu 中的 12 种操作，当指令需要使用到 alu 寄存器中的操作时，只要使线束中对应的 op_’操作’为 1 即可。

源操作数 1 选择方式线束：rs 对应地址的寄存器、PC、sa(inst[10:6])0 扩展，选择指令对应的源操作数 1 时使其对应方式为 1 即可。

源操作数 2 选择方式线束：rt 对应地址的寄存器、立即数 imm(inst[15:0])符号扩展、立即数 imm 0 扩展、32’b8，使用与操作数 1 同理。

源操作数:有 5 种来源方式;当指令为 mflo(mfhi)时,此时的源操作数 rdata 等于 mfdata，正常情况下 mfdata 应该是从 lo(hi)寄存器中读出来的，但若此时的写使能 lo_we(hi_we)为 1，则说明发生了 lo(hi)寄存器的数据相关，因此直接令 mfdata 为要写入的值 lo_wdata(hi_wdata)，代码如下

```
assign mfdata = (inst_mfhi & hi_we)? hi_wdata:
                inst_mfhi? hilo_data:
                 (inst_mflo & lo_we)? lo_wdata:
                 inst_mflo ? hilo_data:
                 32'b0;
```

当指令不为 mflo(mfhi)时，源操作数在通用寄存器不发生数据相关的情况下为从寄存器中读取的数据。而在发生数据相关的情况下,以 EX 段定向为例,当 ex_we 为 1，且 EX 段返回的目的寄存器和当前指令的任何一个源寄存器相同，使用 EX 段返回的数据代替原本从寄存器中读取的数据。MEM 段、WB 段同理，且优先级上 EX > MEM > WB，代码如下所示

```
assign rdata1=(inst_mfhi|inst_mflo)?mfdata:
(rf_waddr_ex==rs && ex_we)?ex_result:
((rf_waddr_mem==rs && mem_we)?mem_result:
((rf_waddr_wb==rs && wb_we)?wb_result:temp_data1));
```

通用寄存器写使能：指令有写入通用寄存器操作时置 1

通用寄存器写入地址线束：rd 对应寄存器、rt 对应寄存器、31 号寄存器，选择指令的写入地址时使其对应方式为 1 即可

(5)跳转模块

这一模块主要完成了跳转使能信号和转移地址的计算。只要通过操作码判定本条指令为跳转指令且满足跳转条件，跳转使能即可置为 1；而对于转移地址，有三种计算方式——分支指令对应的延迟槽指令的 PC 的最高 4 位与立即数 instr_index 左移 2 位后的值拼接、直接取寄存器 rs 中的值、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，其代码如下所示

```
assign br_addr =((inst_jal|inst_j)?{pc_plus_4[31:28],instr_index,2'b0}:
                ((inst_jr|inst_jalr)?temp:
((inst_beq|inst_bne|inst_bgez|inst_bgtz|inst_blez|inst_bltz|inst_bltzal|inst_bgezal) ?
                (pc_plus_4 + {{14{inst[15]}},inst[15:0],2'b0}) : 32'b0)));
```

3.3 EX 段

3.3.1 整体功能说明

完成运算指令的 ALU 操作、乘除法操作，访存指令的地址计算，数据搬移等。

3.3.2 端口介绍

（1）输入端口：

端口名	位宽	作用
clk	1	时钟信号
rst	1	复位信号
stall	`StallBus	暂停气泡
id_to_ex_bus	`ID_TO_EX_WD	ID 段至 EX 段线束

表 3.3.2-1 EX 段输入端口表

（2）输出端口：

端口名	位宽	作用
ex_to_mem_bus	`EX_TO_MEM_WD	EX 段至 MEM 段线束
ex_to_id_bus	38	EX 至 ID 返回信号线束
inst_is_lw	1	是否是 lw 指令
data_sram_en	1	数据内存读使能
data_sram_wen	4	数据内存写使能
data_sram_addr	32	数据内存地址
data_sram_wdata	32	数据内存待写数据
stallreq_for_ex	1	EX 段暂停请求信号
ex_hilo	66	EX 到 ID 段 hilo 线束

表 3.3.2-1 EX 段输出端口表

3.3.3 信号介绍

（1）ex_to_id_bus(EX 至 ID 返回信号)：用于解决数据相关问题。其中，31:0 位为 EX 段计算得到的结果；36:32 位表示写回寄存器的地址；37 位表示写回寄存器使能。当写回寄存器使能信号为高，且写回寄存器的地址与 ID 段需要用到的地址一样时，说明发生了数据相关。需要定向路径将 EX 段的计算结果传至 ID 段，让 ID 段提前拿到结果，以解决数据相关。

（2）inst_is_lw(lw 指令信号)：当 inst_is_lw 信号为高时，表明当前时 lw 指令，传至 ID 段用于进一步判断是否需要暂停。

（3）stallreq_for_ex(EX 段暂停请求信号)：EX 段发给 CTRL 模块的暂停请求信号，当信号为高时表明 EX 段需要暂停（一般由乘除法引起），由 CTRL 模块向 EX 段插入气泡暂停 EX 段。

（4）ex_hilo(EX 段到 ID 段 hi、lo 寄存器相关信号)：EX 段到 ID 段 hi、lo 寄存器相关信号。其中，31:0 位表示待写入 lo 寄存器的值，63:32 位表示待写入 hi 寄存器的值，64 位表示 lo 寄存器写使能信号，65 位表示 hi 寄存器写使能信号。该线束用于控制 lo 和 hi 寄存器的写入。

3.3.4 实例化对象说明

（1）实例化 alu 模块：EX 段实例化了 alu 模块，用于进行加、减、比较大小、与、或、异或、移位等类型的运算，根据输入信号选择运算类型，返回运算结果。

（2）实例化乘法器：EX 段实例化了乘法模块 mul，用于进行 32 位有符号数或无符号数的乘法，返回运算结果。该乘法器为 booth-Wallace 乘法器，可一个周期内完成乘法运算。

（3）实例化除法器：EX 段实例化了除法模块 div，用于进行 32 位有符号数或无符号数的除法，返回运算结果。该除法器基于试商法实现，完成除法运算需要 32 个周期。

（4）实例化复用主要逻辑的综合乘除法器（可选模块）：为完成可选模块，我们设计并完成了复用主要逻辑的综合乘除法器，并在 EX 段实例化。该除法器可进行 32 位有符号数或无符号数的除法或除法，返回运算结果。根据输入的信号选择进行何种移位和加减法，乘法和除法都需要 32 个周期完成。

3.3.4 功能模块说明

EX 段的功能主要有以下三部分：

（1）进行加减、逻辑、移位等运算。首先通过两个选择器分别选择操作数的类型，操作数 1 优先级为 pc 值>sa 字段零扩展>寄存器值，操作数 2 优先级为立即数符号扩展>8>立即数零扩展>寄存器值，代码如下所示：

```
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;
```

然后将信号和操作数送入实例化的 alu 模块，根据信号选择运算类型，输出运算结果。

（2）load 指令和数据搬移指令相关信号的赋值。在 EX 段根据指令相应指令给相应信号赋值，控制指令正确执行，如判断是否是数据搬移指令以确定 EX 段的结果是 alu 的运算结果还是待搬移的数据，代码如下所示。

```
assign move_result=inst_is_move ? rf_rdata1: 32'b0;
assign ex_result = inst_is_move ? move_result : alu_result;
```

（3）进行乘除运算。在 EX 段可进行有符号数和无符号数的乘除法运算，通过实例化的 mul 模块和 div 模块（或自行实现的 mul_div 模块），根据信号选择进行乘法或除法，返回乘法或除法结果。进行除法或自行实现的综合乘除法运算时，需要 32 个周期，乘除法操作未完成时，需要 EX 段向 CTRL 模块发送暂停请求信号，由 CTRL 模块向 EX 段插入气泡以暂停，保证流水线正确进行。

3.4MEM 段

3.4.1 整体功能说明

根据译码结果和EX段的运算结果,对需要访存的指令,进行内存内容操作,包括读、写操作。

3.4.2 端口介绍

(1)输入端口：

端口名	位宽	作用
clk	1	时钟信号
rst	1	复位信号
stall	`StallBus	暂停状态
ex_to_mem_bus	`EX_TO_MEM_WD	跳转总线
data_sram_rdata	32	数据内存读取接口

表 3.4.2-1：MEM 段输入端口表

(2)输出端口：

端口名	位宽	作用
mem_to_wb_bus	`MEM_TO_WB_WD	MEM 段到 WB 段线束
mem_to_id_bus	38	MEM 段到 ID 段线束

表 3.4.2-2：MEM 段输出端口表

3.4.3 信号介绍

(1) rf_we：该信号为寄存器写入使能信号。从线束 ex_to_mem_bus 中取出，最终打包进 mem_to_wb_bus、mem_to_id_bus 线束送入其他流水段。

(2) signal_load：该信号为访存信号。其代表需要访存指令的编号。根据编号的不同，对访存读取的数据进行处理。

3.4.4 功能模块说明

MEM 段的主要功能有以下三部分：

(1)复位处理

在复位时，寄存器 ex_to_mem_bus_r 赋值为`EX_TO_MEM_WD'b0；在复位信号为 0、暂停状态为当前段暂停且下一段不暂停时，寄存器 ex_to_mem_bus_r 赋值为`EX_TO_MEM_WD'b0；在复位信号为 0、暂停状态为当前段不暂停时，寄存器 ex_to_mem_bus_r 连接线束 ex_to_mem_bus。本功能代码如下：

```
always @ (posedge clk) begin
    if (rst) begin
        ex_to_mem_bus_r <= `EX_TO_MEM_WD'b0;
    end
    // else if (flush) begin
    //     ex_to_mem_bus_r <= `EX_TO_MEM_WD'b0;
    // end
    else if (stall[3]==`Stop && stall[4]==`NoStop) begin
        ex_to_mem_bus_r <= `EX_TO_MEM_WD'b0;
    end
    else if (stall[3]==`NoStop) begin
        ex_to_mem_bus_r <= ex_to_mem_bus;
    end
end
```

(2)内存读取与数据确认处理

从寄存器 ex_to_mem_bus_r 中取出 signal_load，接着，利用 mem_result 取出 data_sram_rdata 的结果，随后根据其中不同指令写入确定 regfile 寄存器的 mem_result 数据。此处需要处理的指令有 lb、lbu、lh、lhu。本功能代码如下：

```
assign mem_result=data_sram_rdata;
assign rf_wdata = (signal_load ==3'b001) ? mem_result://lw
(signal_load==3'b010)            &            (ex_result[1:0]==2'b00)?
({{24{mem_result[7]}},mem_result[7:0]}): //lb
(signal_load        ==3'b010)        &        (ex_result[1:0]==2'b01)?
({{24{mem_result[15]}},mem_result[15:8]}): //lb
(signal_load        ==3'b010)        &        (ex_result[1:0]==2'b10)?
({{24{mem_result[23]}},mem_result[23:16]}): //lb
(signal_load        ==3'b010)        &        (ex_result[1:0]==2'b11)?
({{24{mem_result[31]}},mem_result[31:24]}): //lb
```

```
      (signal_load         ==3'b011)         &         (ex_result[1:0]==2'b00)?
({24'b0,mem_result[7:0]}): //lbu
      (signal_load         ==3'b011)         &         (ex_result[1:0]==2'b01)?
({24'b0,mem_result[15:8]}): //lbu
      (signal_load         ==3'b011)         &         (ex_result[1:0]==2'b10)?
({24'b0,mem_result[23:16]}): //lbu
      (signal_load         ==3'b011)         &         (ex_result[1:0]==2'b11)?
({24'b0,mem_result[31:24]}): //lbu
      (signal_load         ==3'b100)         &         (ex_result[1:0]==2'b00)?
({{16{mem_result[15]}},mem_result[15:0]})://lh
      (signal_load         ==3'b100)         &         (ex_result[1:0]==2'b10)?
({{16{mem_result[31]}},mem_result[31:16]})://lh
      (signal_load         ==3'b101)         &         (ex_result[1:0]==2'b00)?
({16'b0,mem_result[15:0]})://lhu
      (signal_load         ==3'b101)         &         (ex_result[1:0]==2'b10)?
({16'b0,mem_result[31:16]})://lhu
     ex_result;
```

(3)整理输出线束

除此之外，需要将 mem_pc,rf_we,rf_waddr,rf_wdata 打包为线束 mem_to_wb_bus 传递给 WB 段；将 rf_we,rf_waddr,rf_wdata 打包为线束 mem_to_id_bus 传回 ID 段，以便解决对应的数据相关。对应代码如下：

```
assign mem_to_wb_bus = {
    mem_pc,      // 41:38
    rf_we,       // 37
    rf_waddr,    // 36:32
    rf_wdata     // 31:0
};
assign mem_to_id_bus = {
    rf_we,       // 37
    rf_waddr,    // 36:32
    rf_wdata     // 31:0
};
```

3.5WB 段

3.5.1 整体功能说明

将运算结果保存到目的寄存器。

3.5.2 端口介绍

(1)输入端口：

端口名	位宽	作用
clk	1	时钟信号
rst	1	复位信号
stall	`StallBus	暂停状态
mem_to_wb_bus	`MEM_TO_WB_WD	MEM 段到 WB 段线束

表 3.5.2-1：WB 段输入端口表

(2)输出端口：

端口名	位宽	作用
wb_to_rf_bus	`WB_TO_RF_WD	WB 段向 regfile 发送的线束
wb_to_id_bus	38	MEM 段到 ID 段线束
debug_wb_pc	32	pc 接口
debug_wb_rf_wen	4	写入寄存器的使能线
debug_wb_rf_wnum	5	写入寄存器的地址
debug_wb_rf_wdata	32	写入寄存器的数据

表 3.5.2-2：MEM 段输出端口表

3.5.3 信号介绍

inst_sram_wen：该信号为寄存器写入使能信号。从线束 ex_to_mem_bus 中取出，最终打包进 mem_to_wb_bus、mem_to_id_bus 线束送入其他流水段。

3.5.4 功能模块说明

WB 段的主要功能如下：

(1)复位处理

在复位时，寄存器 mem_to_wb_bus_r 赋值为 32'hbfbf_fffc；在复位信号为 0、暂停状态为当前段暂停且下一段不暂停时，寄存器 mem_to_wb_bus_r 赋值为 `MEM_TO_WB_WD'b0；在复位信号为 0、暂停状态为当前段不暂停时，寄存器 mem_to_wb_bus_r 连接线束 mem_to_wb_bus。本功能代码如下：

```
always @ (posedge clk) begin
    if (rst) begin
        mem_to_wb_bus_r <= `MEM_TO_WB_WD'b0;
    end
    else if (stall[4]==`Stop && stall[5]==`NoStop) begin
        mem_to_wb_bus_r <= `MEM_TO_WB_WD'b0;
    end
    else if (stall[4]==`NoStop) begin
        mem_to_wb_bus_r <= mem_to_wb_bus;
    end
end
```

(2)整理输出线束

除此之外，需要将 rf_we,rf_waddr,rf_wdata 打包为线束 wb_to_rf_bus 传递给 ID 段，进行写入寄存器操作；将 rf_we,rf_waddr,rf_wdata 打包为线束 wb_to_id_bus 传回 ID 段，以便解决对应的数据相关。对应代码如下：

```
assign wb_to_rf_bus = {
    rf_we,
    rf_waddr,
    rf_wdata
};
assign wb_to_id_bus = {
    rf_we,
    rf_waddr,
    rf_wdata
};
```

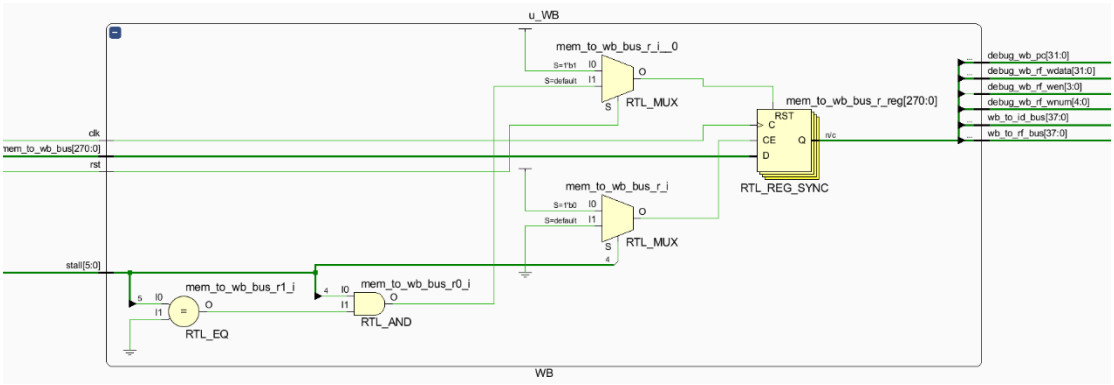

图 3.5.3-1：WB 段结构模型图

3.6 暂停处理

3.6.1 暂停应用情况

暂停应用于定向解决不了的数据相关问题，在 test 文件中比较典型的两种分别如下所示

```
bfc00d58:   8d0a0000    lw  t2,0(t0)
bfc00d5c:   01495026    xor t2,t2,t1

bfc7d7d8:  0109001a    div zero,t0,t1
bfc7d7dc: 0000a812    mflo    s5
```

3.6.2 暂停处理方法

我们小组分别使用了两种不同的暂停处理方法

(1)分别在 EX、ID 段插入气泡，暂停其后面的流水段

上述第一种情况发生时 EX 段的 inst_is_load 信号为 1，ID 段判断条件 EX 段传回来的地址和当前指令源寄存器相同且 inst_is_load 信号为 1，则使 ID 段

的 stallreq 为 1，将 ctrl 中的 stall 数组改成 000111，从而停掉 EX 段及之后的流水段。

上述第二种情况发生时 EX 段的 stallreq_for_ex 信号为 1,，将 ctrl 中的 stall 数组改成 001111，从而停掉 ID 段及之后的流水段。

此外，处理器直接用了存储器当指令的级间寄存器，暂停的时候会有一条指令丢失，因此需要加一个类似于寄存器的器件将这条指令暂时保存下来，而这部分已经在 ID 段的取指令部分详细阐述过，在这里不再赘述。

(2)通过 IF 在 ID 段插入气泡，让前面的流水段重复输出。

上述第一种情况为 load 引起的数据相关，无法直接通过定向路径解决。故需要暂停 ID 段，再通过定向解决数据相关。与第一种处理方法不同，本方法在 ID 段插入气泡，暂停 ID 段，同时让前面的流水段重复输出相同数据。此时，IF 段会向 ID 段重复输出相同指令两次，需要在 ID 段屏蔽掉第一次输出的指令，让它不进行，第二次输入至 ID 段的数据正常进行。CTRL 模块对应代码如下所示：

```
if (stallreq_for_load) begin
    stall[0] = `Stop;
    stall[1] = `Stop;
    stall[2]=`NoStop;
    stall[`StallBus-1:3] = 3'b`Stop;
end
```

4. 可选模块

4.1 上板验证

从 sram 接口一直到 axi 接口，并能通过仿真，可上版验证。
验证时，指示灯跳跃闪烁显示到 40。

4.2 复用主要逻辑的移位乘除法器

4.2.1 端口说明：

（1）输入端口：

端口名	位宽	作用
Rst	1	用于控制复位

Clk	1	时钟信号
sel_mul_div	1	乘除法选择信号
signed_i	1	符号数选择信号
opdata1_i	32	乘除法源操作数 1
opdata2_i	32	乘除法源操作数 2
start_i	1	开始信号
annul_i	1	取消信号

表 4.2.1-1 乘除法器输入端口表

（2）输出端口：

端口名	位宽	作用
result_o	64	乘除法结果
ready_o	1	完成信号

表 4.2.1-2 乘除法器输出端口表

4.2.2 信号说明

（1）sel_mul_div（乘除法选择信号）：用于选择乘除法器进行除法还是除法，以便控制乘除法器进行相应的移位和加减法。

（2）signed_i（符号选择信号）：用于选择进行乘除法操作的是有符号数还是无符号数。如果是有符号数需要先求补码得到原码，用原码进行运算，最后还须对结果求补码才是最终结果。

（3）start_i（开始运算信号）：用于控制乘除法器运算是否开始，当开始信号为高时乘除法器启动运算。

（4）annul_i（取消运算信号）：用于控制乘除法器运算是否取消，当取消信号为高时乘除法器取消运算。

（5）ready_o（运算完成信号）：当运算完成信号为高时表明乘除法运算已完成，EX 段的暂停结束，流水线继续进行。

4.2.3 功能模块说明

实现了复用主要逻辑的移位综合乘除法器，整合乘除法，用于进行 32 位二进制数的乘除法，包括无符号数和有符号的乘除法。

4.2.4 算法流程

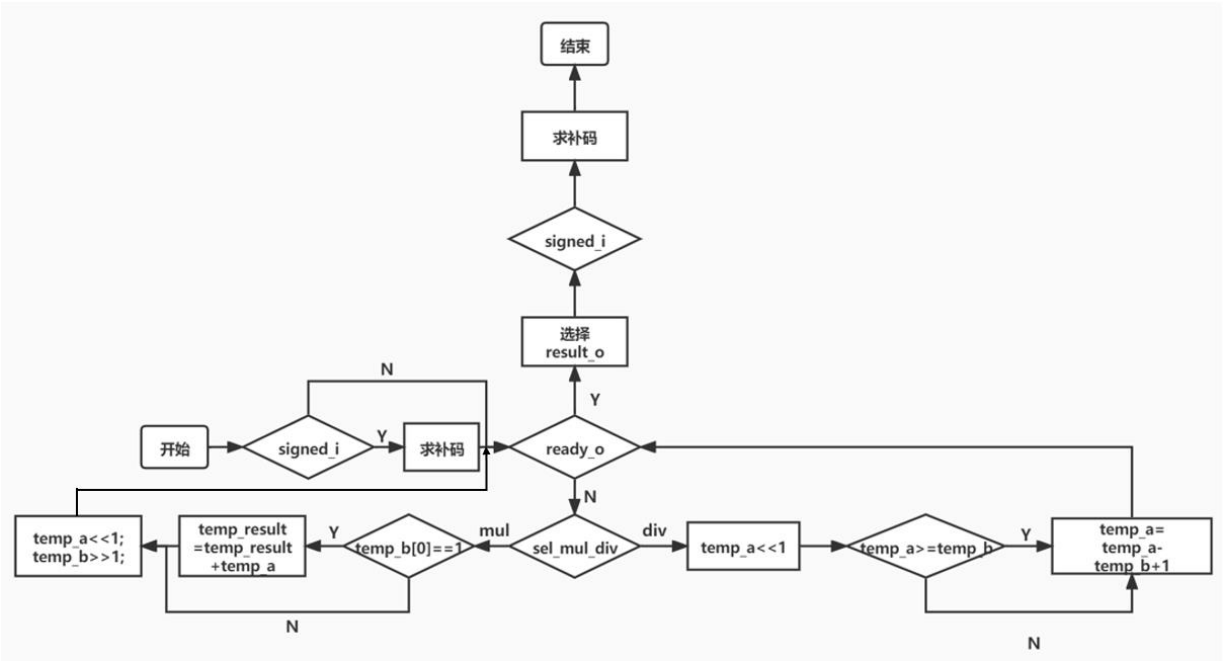

图 4.2.4-1　算法流程图

自行实现的复用主要逻辑的综合乘除法器算法流程图如上所示，具体流程：

1、判断是否为有符号数乘除法，是执行 2，否执行 3；

2、对两个操作数求补码，得到原码 temp_a,temp_b；

3、判断乘除法是否结束，结束条件为移位 32 次，是执行 13，否执行 4；

4、判断进行乘法或除法，若进行乘法执行 5，否则执行 9；

5、判断 temp_b 的最低位是否为 1，是执行 6，否则执行 7；

6、temp_result = temp_result + temp_a；

7、temp_a 左移 1 位，temp_b 右移 1 位；

8、返回 3；

9、操作数 1(temp_a)左移 1 位；

10、判断 temp_a 是否大于 temp_b，是执行 11，否则执行 12；

11、temp_a = temp_a - temp_b + 1；

12、返回 3；

13、选择 result_o，如果是乘法，result_o = temp_result；如果是除法,result_o = temp_a；

14、判断是否为有符号数乘除法，是执行 15，否结束；

15、求 result_o 的补码，结束。

5. 实验总结

5.1 严洁 实验总结

本次计算机系统实验让我增加了实践经验，通过实践巩固课内所学知识，理论与实践相互结合，加深了我对计算机体系结构的了解。通过解决功能测试点中的数据相关问题，我更加深入地理解了定向路径的使用和作用；通过解决 load 导致的数据相关，我学会了通过插入气泡解决定向路径无法解决的数据相关问题以使流水线正确工作；通过实现移位综合乘除法，我提高了我对 CPU 性能等各方面的理解；通过添加指令以通过功能测试点，我更加深入地了解了 MIPS 的指令系统设计以及各指令的工作原理。本次实验还提高了我的自学能力，通过自学 verilog、vivado 等，我掌握了更加适合自己的学习方法，提高了我的自学能力。此外，本次实验还提高了我发现问题和解决问题的能力，从一开始的不会看波形图只会猜错试错，到后期能够熟练运用波形图 debug，分析、寻找和解决错误。

5.2 王馨霄 实验总结

这次实验给我带来的感受是：实现这个五级流水的过程中，我不断复习课堂中的理论知识，并在实践中不断印证它，这加深了我对理论知识的理解。而实现过程中也有一些和理论中并不完全一样的方法，这也让我对工程思想也有了一定了解。

针对本次实验，我的改进意见是：实验的小组合作形式还需商榷。这一实验是纯粹串行性质的，后面功能点的完成是在前面功能点的基础上进行的，这就导致很难分工。而像我们组这样每人都从头开始自己完成一份，虽然我认为这对我们掌握知识是有好处的，但确实也让小组合作失去了一定的意义。

5.3 魏钞迪 实验总结

通过本次实验，我初次接触 verilog 语言，并且认识到其与已经接触到的所有编程语言都有所不同。verilog 语言最特殊的一点是具有同时性。这让我在理解本实验程序时产生了不小的困惑。多亏，助教学长与小组成员的帮助，我才能够进一步理解并将实验进行下去。除此之外，本次实验让我深刻地理解了 CPU 五级流水结构的原理与性能，帮助我融会贯通课上所学知识且付诸实践。切实体会到了暂停处理、寄存器读写等等操作的实践逻辑。本次实验虽然是小组合作，但是小组合作过程略显困难。试验进度难以同时进行，只能小组成员接力任务甚至需要先理解前一组员代码的基础上接力。学生认为这样的合作方式需要改进。

6. 参考资料

[1] 雷思磊．自己动手做 cpu［M］．电子工业出版社，2014.

[2] 张晨曦，王志英，张春元等．计算机体系结构［M］．高等教育出版社，2000.