实验报告成绩：	成绩评定日期：

2024 ~ 2025 学年秋季学期

A3705060050《计算机系统》必修课

课程实验报告

班级： 人工智能未来实验班2201

组长： 韩翼聪

组员： 祁记、艾学桐

报告日期：2024.1.2

目录

一、1、小组成员工作量划分

姓名	完成任务点	总任务量占比
韩翼聪	暂停的部分实现、乘除法器的连接、 一部分指令的添加	30%
祁记	数据相关的解决、暂停的部分实现、 一部分指令的添加、自制乘法器的实现	40%
艾学桐	一部分指令的添加、报告的撰写	30%

二、 总体设计

我们的流水线 CPU 设计主要由七个模块组成，分别为 IF 模块、ID 模块、EX 模块、MEM 模块、WB 模块、CTRL 模块，以及 regfile 模块。总体具有 32 个 32 位的整数寄存器，并且采用大端模式储存，具有 32bit 数据，地址总线宽度。我们的流水线 CPU 可以完成实验课程所要求的 64 个测试点， 而且我们设计了自制的 32 周期的 32 位移位乘法器，并且已经通过了上板验证。

取指阶段（IF）：从内存中获取指令，并控制指令延迟槽和跳转指令，同时确定下一条指令的地址。

译码阶段（ID）：对指令进行译码，读取通用寄存器的值，并控制hilo寄存器的读写。若指令含有立即数，则设置立即数及其扩展方式；选择合适的ALU运算单元，判断转移指令的跳转条件，并计算新的指令地址。

执行阶段（EX）：根据译码阶段的操作数和ALU类型，进行运算。如果是加载或存储指令，还会计算内存地址并设置内存交互方式。

访存阶段（MEM）：判断是否将ALU的结果或从内存中读取的值写回寄存器，并将数据传递到回写阶段。

回写阶段（WB）：将运算结果存入CPU的32个32位寄存器。在提高效率的设计中，回写寄存器的值和地址会提前传给译码阶段，在译码阶段直接进行寄存器写入。

控制阶段（CTRL）：负责流水线的暂停、清除等控制操作。

实验环境：使用 VS Code 作为 Vivado2019.2 的默认代码编辑器，使用 git 完成小组内部的协同开发和代码同步。

三、流水线各个阶段的说明：

1、IF模块

IF模块功能：取指令，并控制指令延迟槽和跳转指令

IF模块接口：

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	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
5	if_to_id_bus	33	输出	IF段发给ID段的数据
6	inst_sram_en	1	输出	指令寄存器的读写使能信号
7	inst_sram_wen	4	输出	指令寄存器的写使能信号
8	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
9	inst_sram_wdata	32	输出	指令寄存器的数据，用来存放数据

IF模块细述：

IF段接收时钟信号和复位信号，若复位信号为真，则将PC值置为复位值。它还接收来自CTRL段的stall信号，当流水线暂停时，IF段通过识别stall信号暂停指令延迟槽，使得PC保持当前值，暂停一个时钟周期。br_bus是来自ID段的跳转信号，包含跳转指令的条件和目标地址。如果需要跳转，IF段会更新next_pc为目标地址。正常情况下，IF段将reg_pc设为当前的next_pc，并将next_pc加4，之后将reg_pc地址传送给指令内存，从中获取对应的指令并传给ID段。

2、ID 模块

ID 模块功能：对指令进行译码， 并将译码结果传给 EX 段， 同时与寄存器进行交互，实现寄存器的读写，处理数据相关。

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	ex_is_load	1	输入	判断上一条指令是否是加载指令
5	stallreq	1	输出	控制暂停
6	if_to_id_bus	33	输入	IF 段发给 ID 段的数据
7	inst_sram_rdata	32	输入	当前指令地址中储存的值
8	wb_to_rf_bus	38	输入	WB 段传给 ID 段要写入寄存器的值
9	ex_to_id	38	输入	EX 段传给 ID 段的数据，用来判断数据相关
10	mem_to_id	38	输入	MEM 段传给 ID 段的数据，用来判断数据相关
11	wb_to_id	38	输入	WB 段传给 ID 段的数据，用来判断数据相关
12	hilo_ex_to_id	66	输入	EX 传给 ID 段要写入乘除法寄存器的值
13	id_to_ex_bus	169	输出	ID 段传给 EX 段的数据
14	br_bus	33	输出	ID 段传给 IF 段的数据，用来判断下一条指令的地址
15	stallreq_from_id	1	输出	从 ID 段发出的暂停信号

ID 模块细述：

ID 段分为几段：

第一段： 暂停判断与气泡处理：ID段根据CTRL模块的stall信号判断是否暂停流水线。若为NoStop，流水线继续，正常接收IF段传来的if_to_id_bus。若为Stop，表示访存冲突，ID段暂停一周期，保存当前指令（inst）直到下周期再使用。

第二段： 指令译码：根据指令特征，ID段进行译码，激活相应指令的标识符，并读取通用寄存器中的数据。ALU操作符根据指令类型赋值，并设置写使能信号和目标寄存器。跳转指令根据条件设置跳转地址。

第三阶段： 操作数来源选择：ID段设置操作数来源。sel_alu_src1选择第一个操作数来源（如寄存器、PC或立即数），sel_alu_src2选择第二个操作数来源（如寄存器、符号扩展或立即数）。

第四阶段：数据传递给后续阶段：ID段将译码结果、ALU操作符、操作数来源等信号传递给EX段，同时将跳转信息传递给IF段。

在**寄存器模块**中，32个通用寄存器和乘除法的hi、lo寄存器用于数据存取。读取时根据控制信号hi_r和lo_r判断是否输出hi或lo寄存器的值。同时，通过寄存器地址raddr1、raddr2读取相应寄存器的值，若地址为0，则返回0。

传给 IF 段，告诉 IF 段目前指令是否为跳转指 令，以及要跳转的地址。

```
assign id_to_ex_bus = {
    data_ram_readen,//168:165
    inst_mthi,      //164
    inst_mtlo,      //163
    inst_multu,     //162
    inst_mult,      //161
    inst_divu,      //160
    inst_div,       //159
    id_pc,          // 158:127
    inst,           // 126:95
    alu_op,         // 94:83
    sel_alu_src1,   // 82:80
    sel_alu_src2,   // 79:76
    data_ram_en,    // 75
    data_ram_wen,   // 74:71
    rf_we,          // 70
    rf_waddr,       // 69:65
    sel_rf_res,     // 64
    rdata11,         // 63:32
    rdata22          // 31:0
};
```

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

序号	接口名	宽度	输入/输出	作用
1	clk	1	输入	时钟信号
2	raddr1	5	输入	请求读取的第一个数的地址
3	rdata1	32	输出	读取出的第一个数的值
4	raddr2	5	输入	请求读取的第二个数的地址
5	rdata2	32	输出	读取出的第二个数的值
6	we	1	输入	是否要写入寄存器的写使能信号
7	waddr	5	输入	要写入的地址
8	wdata	32	输入	要写入寄存器的值
9	hi_r	1	输入	是否要读取 hi 寄存器的值的读使能信号
10	hi_we	1	输入	是否要写入 hi 寄存器的写使能信号
11	hi_data	32	输入	要写入 hi 寄存器的值
12	lo_r	1	输入	是否要读取 lo 寄存器的值的读使能信号
13	lo_we	1	输入	是否要写入 lo 寄存器的写使能信号
14	lo_data	32	输入	要写入 lo 寄存器的值
15	hilo_data	32	输出	从 hilo 寄存器中读取出来的值

regfile模块包含32个32位通用寄存器和1对乘除法的hi与lo寄存器。对于乘除法寄存器的读取，hi和lo寄存器不会同时输出，只会输出一个。读取时，若hi_r为1，输出hi寄存器的值；若lo_r为1，输出lo寄存器的值；若两者都为0，输出0。

对于rs和rt寄存器的值，若raddr1为0，则rdata1为0；否则，rdata1为raddr1对应寄存器的值。同理，若raddr2为0，则rdata2为0，否则为raddr2对应寄存器的值。

3、EX 模块

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。

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	EX 发出的是否暂停的信号
12	ex_is_load	1	输出	EX 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

EX 模块细述：

定义如下变量

data_ram_readen,inst_mthi,inst_mtlo,inst_multu,inst_mult,inst_divu,inst_div,ex_pc,inst,alu_op,sel_alu_src1,sel_alu_src2,data_ram_en,data_ram_wen,rf_we,rf_waddr,sel_rf_res,rf_rdata1,rf_rdata2 将 ID 段传给 EX 段 id_to_ex_bus_r 复制给相应变量。

在EX段处理流水线中，涉及到多个指令类型的判断和相应的操作，主要包括如下简化步骤：

传递变量：ID段将各类控制信号（如data_ram_readen、inst_mthi、inst_mtlo等）通过id_to_ex_bus_r传递到EX段。

判断指令类型：

判断是否为LW指令（inst[31:26] == 6'b10_0011），若是，设置ex_is_load为1；否则为0。

计算操作数：

1. 立即数符号扩展：{{16{inst[15]}}, inst[15:0]}。
2. 立即数零扩展：{16'b0, inst[15:0]}。
3. sa零扩展：{27'b0, inst[10:6]}。

根据sel_alu_src1和sel_alu_src2，设置alu_src1和alu_src2为相应的操作数。

ALU运算：调用ALU模块计算两个操作数，并将计算结果ex_result传出。

内存操作：

1. 判断是否为sb指令（data_ram_readen == 4'b0101），若是，根据ex_result[1:0]设置内存写使能。
2. 如果是sh指令（data_ram_readen == 4'b0111），根据ex_result[1:0]设置内存写使能。

写内存数据：根据data_sram_wen的值决定要写入内存的数据来源。对于sb和sh指令，分别根据地址偏移确定写入的位置。

乘除法指令的处理：

1. 乘法指令：判断指令是否为mul或multu，然后设置mul_ready_i和stallreq_for_mul。若乘法器空闲，传入操作数开始计算，并暂停流水线，等乘法结束后恢复流水线。
2. 除法指令：处理与乘法类似，判断指令类型并设置stallreq_for_div，暂停流水线直到除法完成。

计算结果存储在hilo寄存器中，高位存入hi寄存器，低位存入lo寄存器。传值给ID段：

1. 将内存操作相关信号、ALU结果、写寄存器信号等传递给ID段。
2. 将hi和lo寄存器的写使能信号及数据传递给ID段，确保乘除法器的结果被正确写入。

4、MEM 模块

MEM 模块功能：读取内存中相应地址的值（其实是在 EX 到 MEM 的过程中读取的），根据当前指令，确定要写入寄存器的值。

MEM 模块接口：

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	ex_to_mem_bus	80	输入	EX 传给 MEM 段的数据
5	data_sram_rdata	32	输入	从内存中读出来要写入寄存器的值
6	mem_to_id	38	输出	MEM 传给 ID 段的数据
7	mem_to_wb_bus	70	输出	MEM 传给 WB 段的数据

MEM 模块描述：

在访存（MEM）阶段，主要进行以下操作：

接收传输的信号：

1. 从执行阶段（EX）接收信号：dat
   mem_pc、rf_we、rf_waddr、ex_r
   需要进行内存读写以及需要写入

```
assign {
    data_ram_readen,//79:76
    mem_pc,         // 75:44
    data_ram_en,    // 43
    data_ram_wen,   // 42:39
    sel_rf_res,     // 38
    rf_we,          // 37
    rf_waddr,       // 36:32
    ex_result       // 31:0
} =  ex_to_mem_bus_r;
```

及是否

判断是否进行内存读取：

1. **LW指令**：若是LW指令（data_ram_readen为6'b10_0011），将从内存读取的值作为写入寄存器的数据。

处理特定指令类型的读取：

1. **LB指令**：若是LB指令，根据地址最后两位决定从内存读取哪个字节，并进行符号扩展：
    1. 2'b00：读取第一个字节，符号扩展。
    2. 2'b01：读取第二个字节，符号扩展。

3. 2'b10：读取第三个字节，符号扩展。
4. 2'b11：读取第四个字节，符号扩展。

2. **LBU指令**：若是LBU指令，根据地址最后两位决定从内存读取哪个字节，并进行零扩展：

    1. 2'b00：读取第一个字节，零扩展。
    2. 2'b01：读取第二个字节，零扩展。
    3. 2'b10：读取第三个字节，零扩展。
    4. 2'b11：读取第四个字节，零扩展。

3. **LH指令**：若是LH指令，根据地址最后两位决定从内存读取哪些字节，并进行符号扩展：
    1. 2'b00：读取前两个字节，符号扩展。
    2. 2'b10：读取后两个字节，符号扩展。
4. **LHU指令**：若是LHU指令，根据地址最后两位决定从内存读取哪些字节，并进行零扩展：

    1. 2'b00：读取前两个字节，零扩展。
    2. 2'b10：读取后两个字节，零扩展。

默认情况：若不是上述任何指令（即既不是LB、LBU、LH、LHU、LW等），则直接使用从执行

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

与写入寄存器的数据。

传值给 WB 阶段：

将当前的 PC 值，以及是否要写入寄存 器，以及要写入寄存器的值还有数据发给回写段。

传值给 ID 段：将当前的 PC 值，以及是否要写入寄存 器，以及要写入寄存器的值还有数据发给 译码段，用来判断是否发生数据相关。

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

5、WB 模块

WB 模块功能：将结果写入寄存器，这一段实际是在 ID 段调用 regfile 模块实现的。

WB 模块接口：

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	mem_to_wb_bus	70	输入	MEM 传给 WB 的数据
5	wb_to_rf_bus	38	输出	WB 传给 rf 的数据
6	wb_to_id	38	输出	WB 传给 ID 的数据
7	debug_wb_pc	32	输出	用来 debug 的 pc 值
8	debug_wb_rf_wen	4	输出	用来 debug 的写使能信号
9	debug_wb_rf_wnum	5	输出	用来 debug 的写寄存器地址
10	debug_wb_rf_wdata	32	输出	用来 debug 的写寄存器数据

WB 模块细述：

接受数据：

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

传值给 rf 寄存器：

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

接受从 MEM 传过来的 PC 值，以及是否要 写入寄存器，以及要写入寄存器的值还有数据。
将当前的 PC 值，以及是否要写入寄存 器，以及要写入寄存器的值还有数据发给 rf 寄存器模块。

6、CTRL 模块：

CTRL 模块功能： 接收各段传递过来的流水线请求信号，从而控制流水 线各阶段的运行。 （在本实验中目前为止只有俩个会发送请求信号，译码阶段和 执行阶段，取值和访存阶段没有暂停请求，应为这列个阶段的操作都可以在一 个周期内完成）

CTRL 模块原理：

假设位于流水线第 n 阶段的指令需要多个周期，进而请求流水线暂停，那 么需要保持取指令地址 PC 不变，同时保持流水线第 n 阶段、第 n 阶段之前的各 个阶段的寄存器保持不变，而第 n 阶段后面的指令继续运行。比如：流水线执 行阶段请求暂停流水线，那么保持 PC 不变，同时保持取值，译码，执行阶段的 寄存器不变，但是可以允许访存、回写的指令继续运行。

CTRL 模块接口：

序号	接口名	宽度	输入/输出	作用
1	rst	1	输入	复位信号
2	stallreq_from_ex	1	输入	处于译码阶段的指令是否请求流水线暂停
3	stallreq_from_id	1	输入	处于执行阶段的指令是否请求流水线暂停
4	stall	6	输出	暂停流水线控制信号

输出暂停信号：
stall[0]为 1 表示没有暂停
stall[1]为 1 stall[2]为 1 stall[3]为 1 stall[4]为 1

if 段暂停

id 段暂停

ex 段暂停

mem 段暂停

stall[5]为 1 wb 段暂停

判断处于译码阶段的指令是否请求流水线暂停，如果请求暂停，那么就把 流水线暂停控制信号赋值为 6'b001111，表示取值阶段暂停，译码阶段暂停，执行阶段暂停，访存阶段不暂停，回写阶段不暂停。

判断处于译码阶段的指令是否请求流水线暂停，如果请求暂停，那么就把流水线暂停控制信号赋值为 6'b000111，表示取值阶段暂停，译码阶段暂停， 执行阶段暂停，访存阶段不暂停，回写阶段不暂停。

如果都不暂停将流水线控制信号赋值为 6'b000000，表示不暂停。

在对应的 IF、ID、EX、MEM、WB 段中都有对应的读取 stall 值控制对应模 块是否暂停的部分。

7、MUL模块：

MUL 模块功能：实现两个 32 位有符号或者无符号数的乘法，并返回一个 64 位的结果。

MUL 模块原理：先把被乘数扩展成 64 位，再左移 32 位，每一次左移之前都要判断乘数最低位是否为 1，为 1 则把那一步的被乘数加到结果里（result），为 0 则不做处理，于此同时将乘数右移一位。

MUL 模块接口：

序号	接口名	宽度	输入输出	作用
1	rst	1	输入	复位信号，高电平有效
2	clk	1	输入	时钟信号
3	signed_mul_i	1	输入	是否为有符号乘法，为 1 表示有符号乘法
4	a_o	32	输入	被乘数
5	b_o	32	输入	乘数
6	start_i	1	输入	是否开始乘法运算
7	result_o	64	输出	乘法运算结果
8	ready_o	1	输出	乘法运算是否结束

MUL 模块总述：

MUL 模块的主要部分是一个状态机，由三个状态。

MulFree：乘法模块空闲

MulOn：乘法运算进行中

MulEnd：乘法运算结

复位的时候， MUL 模块处于 MULFREE 状态，当输入信号 start_i 为 `MulStart 时，表示乘法操作开始。

进入 MulOn 状态，使用移位乘法，经过 32 个周期，得到乘法结果，然后进 入 MulEnd 模块，并通知 EX 模块得到乘法运算结果。

reg [5:0]i3//进行到第几位

reg [31:9]temp_opa,temp_opb;

表示被乘数以及乘数

```
`MulOn: begin                //乘法运算
        if(i != 6'b100000) begin
            if(temp_opb[0]==1'b1) begin
                pv <= pv + ap;
                ap <= {ap[62:0],1'b0};
                temp_opb <= {1'b0,temp_opb[31:1]};
            end
            else begin
                ap <= {ap[62:0],1'b0};
                temp_opb <= {1'b0,temp_opb[31:1]};
            end
            i <= i + 1;
        end
        else begin
            if ((signed_mul_i == 1'b1) && ((a_o[31] ^ b_o[31])
                pv <= ~pv + 1;
            end
            state <= `MulEnd;
            i <= 6'b00_0000;
        end
    `MulFree: begin          //乘法器空闲
        if (start_i== `MulStart) begin
end
            state <= `MulOn;
            i <= 6'b00_0000;
            if(signed_mul_i == 1'b1 && a_o[31] == 1'b1) begin
                temp_opa = ~a_o + 1;
            end else begin
                temp_opa = a_o;
            end
            if(signed_mul_i == 1'b1 && b_o[31] == 1'b1 ) begin
                    temp_opb = ~b_o + 1;
            end else begin
                temp_opb = b_o;
            end
            ap <= {32'b0,temp_opa};
            ready_o <= `MulResultNotReady;
            result_o <= {`ZeroWord, `ZeroWord};
            pv <= 64'b0;
        end
    end
```

MulFree：开始乘法运算，如果是 有符号乘法，且被除数 或者乘数为负数，那么 对被除数或者乘数为负 数取补码，被乘数保存 到 ap 的低 32 位。同时 将乘法器结果置为 0,乘 法结果中间值 pv 置为 0。

MulOn：乘法运行状态：

如果 i！=32，如 果乘数的最低位 位 1，那么将 pv 加上一个 ap，同 时将 ap 左移一 位，乘数右移一 位；如果乘数最 低为为零，将 ap 左移一位，乘数 右移一位。 然后将 i+1。 如果 i==32，表示 移位乘法结束。如果似乎有符号乘法，且乘数与被乘数一正一负，将 pv 取补码。

```
    `MulEnd: begin          //乘法结束
        result_o <= pv;
        ready_o <= `MulResultReady;
        if (start_i == `MulStop) begin
            state <= `MulFree;
            ready_o <= `MulResultNotReady;
            result_o <= {`ZeroWord, `ZeroWord};
        end
    end
```

MulEnd：

乘法运算结束,将 pv 赋 值给乘法器结果，结果 设置位可获取状态。同时将乘法器设置为空闲 状态，并赋为初始值。

四、组员感受以及改进意见

韩翼聪：在这次实验中，我将课堂所学应用于实践，深入理解了流水线的整体运行。在编写流水线CPU的过程中，由于缺乏经验和自行设计的部分，遇到了不少困难，如数据相关问题、流水线暂停实现和内存读写的处理等。通过查阅资料和与队友交流，逐步解决了这些问题。特别是在流水线暂停的设计上，我最初对其原理理解不够，导致程序出现bug。后来通过查阅资料并反复调试，才找出问题并进行修正，积累了宝贵的经验。

建议：在实验初期，由于我们缺乏Verilog编程经验，面对复杂模块时感到迷茫。如果能在实验前进行更多的Verilog编程训练，或者在实验开始时提供更详细的文档说明，将有助于更快上手。

祁记：在这次实验中，我深刻体会到“罗马非一日建成”这句话。外表庞大的处理器，也是在一步步构建中完成的。我们不应被其复杂性吓倒，从最简单的地方入手，逐渐增加功能和指令，一行一行代码地书写，脚踏实地，最终会发现自己已经走得很远，克服了许多困难，完成了很多设计。通过这次实验，我对CPU有了更深入的了解，收获颇丰。

建议：希望明年实验能先讲解Verilog的基础，尤其是时序逻辑和组合逻辑，这样大家可以更快上手，避免像今年那样花费较多时间适应。

艾学桐：这次实验加深了我对流水线的理解，并让我更深刻体会到团队合作和模块化设计的重要性。在有限时间内完成任务，必须依赖团队合作，并通过合理的模块化设计来高效分工。总体来说，这次课程设计增强了我的理论知识，提升了编码能力，也让我在团队合作中积累了宝贵经验，受益匪浅。

建议：希望明年实验能先讲解Verilog的基础，不然自学得花费不少时间去配置软件拖慢了很多进度。