成绩评定日期：

实验报告成绩：

2024～2025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2201

组长：王烁天

组员：田人溥、郑一夫

报告日期：2025.1.5

目录

[1. 实验设计 3](#_Toc19256)

[1.1小组成员工作量划分 3](#_Toc7172)

[1.2 总体设计 3](#_Toc32491)

[1.3 运行环境及工具 5](#_Toc19652)

[2. 流水线各个阶段的说明 5](#_Toc23612)

[2.1 IF模块 5](#_Toc27838)

[2.2 ID模块 6](#_Toc16552)

[2.3 EX模块 11](#_Toc11788)

[2.4 MEM模块 13](#_Toc32299)

[2.5 WB模块 15](#_Toc2519)

[2.6 CTRL模块 16](#_Toc25572)

[2.7 HILO寄存器模块 18](#_Toc19197)

[3. 实验感受及建议 19](#_Toc12453)

[3.1 王烁天部分 19](#_Toc32453)

[3.2 田人溥部分 20](#_Toc26449)

[3.3 郑一夫部分 20](#_Toc1109)

[4. 参考资料 21](#_Toc31488)

1. **实验设计**

**1.1小组成员工作量划分**

|  |  |  |
| --- | --- | --- |
| 姓名 | 任务分工 | 任务量占比 |
| 王烁天 | 添加算术运算、数据移动、逻辑、跳转、访存指令 | 50% |
| 田人溥 | 主要负责在流水线中添加stall相关指令，参与实验报告的编写 | 25% |
| 郑一夫 | 主要负责在流水线中添加hilo相关指令，参与实验报告的编写 | 25% |

**1.3 运行环境及工具**

运行环境：本地电脑。 FPGA的Family为Artix 7，Package为fbg676，型号为xc7a200tfbg676-2（和PPT教程一致）。

编程工具：使用VSCode编写代码，使用Vivado 模拟仿真，使用git进行版本管理，使用GitHub搭建项目仓库。

**1.2 总体设计**

项目包括IF.v，ID.v，EX.v，MEM.v，WB.v，hi\_lo\_reg.v，mycpu\_core.v，mycpu\_top.v，这部分搭建了一条流水线的基本框架；及位于/lib目录下的alu.v，decoder\_5\_32.v，decoder\_6\_64.v，defines.vh，div.v，mmu.v，regfile.v，这部分构建了ALU和寄存器，定义了包含总线宽度信息在内的头文件；及位于/lib/mul目录下的add.v，fa.v，mul.v，这部分实现了乘法的运算。

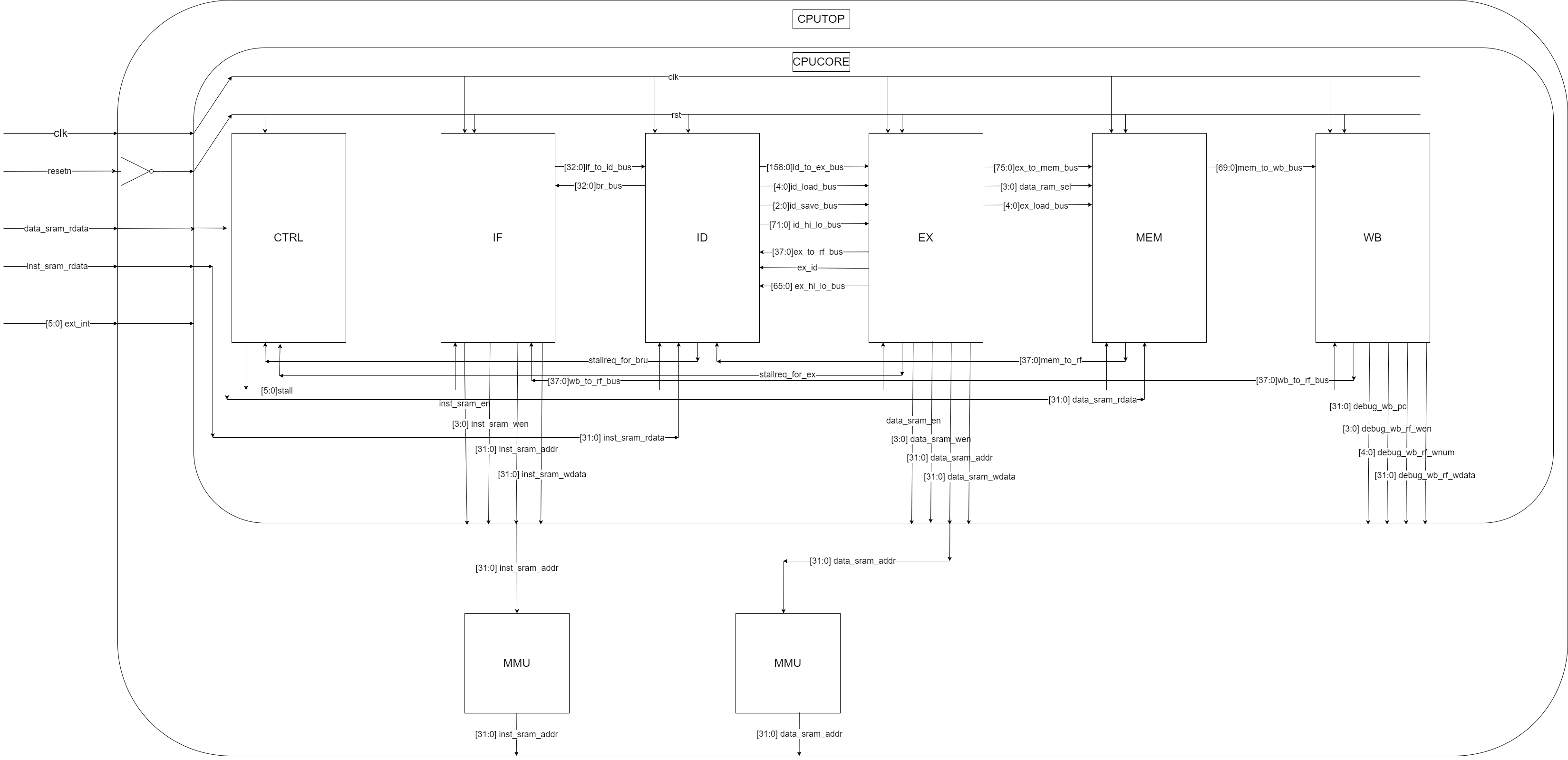
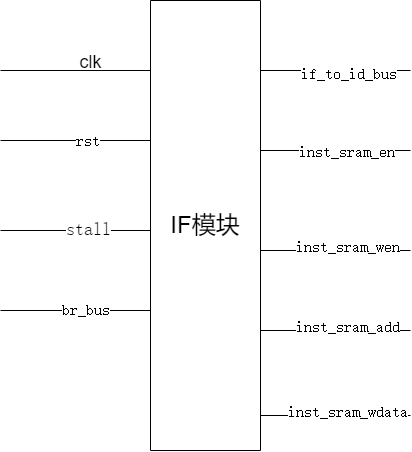
****

图 1 CPU流水线示意图

1. **流水线各个阶段的说明**

**2.1 IF模块**

**整体说明：**取指令，主要负责处理分支延迟槽和跳转指令。该模块的接口如下表所示。

**功能说明：**

复位与暂停：

当复位信号 (rst) 置为真时，pc\_reg 被复位到初始值。

判断暂停信号 stall：若 stall 的对应位为 1，则暂停取指，让下一条指令的 pc 不再更新（保持当前值）。

跳转判断：

判断 br\_bus，若其指示需要跳转，则将 next\_pc 设为 br\_bus 中的跳转地址；否则，next\_pc 等于当前 pc\_reg + 4。

最终，将 next\_pc 写入 pc\_reg，并将 pc\_reg 的值输出给指令存储器，从而获取该地址对应的指令。

与 ID 段交互：

通过 if\_to\_id\_bus 将当前时刻的 pc\_reg 及其他相关信息传递给 ID 段。

表 1 IF模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 暂停信号，控制指令是否暂停 |
| 4 | br\_bus | 33 | 输入 | 分支跳转信号，控制延迟槽是否跳转 |
| 5 | if\_to\_id\_bus | 33 | 输出 | IF段到ID段的数据总线 |
| 6 | inst\_sram\_en  inst\_sram\_en  \_en | 1 | 输出 | 读写使能信号 |
| 7 | inst\_sram\_wen | 4 | 输出 | 写使能信号 |
| 8 | inst\_sram\_addr | 32 | 输出 | 存放指令寄存器的地址 |
| 9 | inst\_sram\_wdata | 32 | 输出 | 存放指令寄存器的数据 |

**2.2 ID模块**

**整体说明：**

对指令进行译码，将结果传给EX段，实现寄存器读写，处理数据相关。接口下图所示。

表 2 ID模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | 暂停信号，控制指令是否暂停 |
| 4 | stallreq | 1 | 输出 | 暂停请求信号 |
| 5 | if\_to\_id\_bus | 33 | 输入 | IF段到ID段的数据总线 |
| 6 | inst\_sram\_rdata | 1 | 输入 | 读写使能信号 |
| 7 | ex\_id | 1 | 输入 | 写使能信号 |
| 8 | wb\_to\_rf\_bus | 38 | 输入 | WB段存放进寄存器的数据 |
| 9 | ex\_to\_rf\_bus | 38 | 输入 | EX段存放进寄存器的数据 |
| 10 | mem\_to\_rf\_bus | 38 | 输入 | MEM段存放进寄存器的数据 |
| 11 | ex\_hi\_lo\_bus | 66 | 输入 | EX段存放进hilo寄存器的数据的总线 |
| 12 | id\_hi\_lo\_bus | 72 | 输出 | ID段存放进hilo寄存器的数据的总线 |
| 13 | id\_load\_bus | 5 | 输出 | ID段执行load命令的数据总线 |
| 14 | id\_save\_bus | 3 | 输出 | ID段执行save命令的数据总线 |
| 15 | stallreq\_for\_bru | 1 | 输出 | 执行load命令时的暂停请求 |
| 16 | id\_to\_ex\_bus | 159 | 输出 | ID段到EX段的数据总线 |
| 17 | br\_bus | 33 | 输出 | 分支跳转信号，控制延迟槽是否跳转 |

**功能说明：**

**第一部分 流水线暂停的判断与实现**

ID 段收到来自CTRL模块的stall值后，会判断stall的值，当对应ID段的部分的stall的值为0时，也就说明没有流水线暂停，那就将将IF段传给ID段的if\_to\_id\_bus正常地赋值给if\_to\_id\_bus\_r，然后就可以进行接下来的译码、取操作数的部分。但是如果判断到 stall 的值的对应的 ID 段的部分是1，也就是说此刻发生了访存冲突，需要读取的寄存器中的值还没有获得，还需要在下一个周期才可以从内存中读取出来，且无法通过数据前递解决，则现在需要对流水线的 ID 段进行暂停一个周期，在下个周期获得到需要读取的值后再发给 ID 段。当判断到暂停后就将 if\_to\_id\_bus\_r 置为 0，本周期停止，下个周期再恢复正常。由于if\_to\_id\_bus不包含指令值，指令的值即 inst 值是在 ID 段时根据上个周期的IF段中的pc值从内存中读取到的,是直接从内存获取的，因此inst 值并没有被置为 0，因此还需要把当前时刻的 inst 值保存一个周期，下 一个周期再使用当前周期的inst值，这样才能保证 ID 段和之后所有部分的指令的pc值和inst值是相互匹配的。

**第二部分 指令的译码**

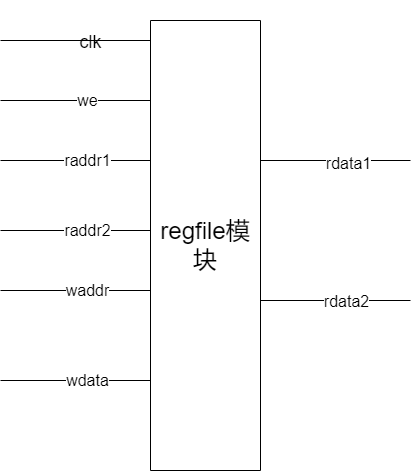
一般指令：先依据指令中的特征字段区分指令，同时激活相应的指令对应的 inst\_\*\*变量，表示是哪一条指令。根据译码结果，读取通过regfile模块读取地址为 rs（inst[25;21]）以及地址为rt(inst[20:16]的通用寄存器,得到rdata1以及rdata2，并且通过判断是否发生数据相关，从而更改rdata1以及 rdata2的值。同时分析要执行的运算，给对应的ALU标识符赋值，其中，0表示该条指令不采用该 ALU，1 表示采用该ALU，同时将所有的 ALU 标识符组合起来成为 alu\_op，alu\_op为十二位宽，代表 16 种不同的 ALU，并且作为传入 EX 段的一部分。要写入的目的寄存器。rf\_we 代表写使能信号，表示该条指令是否用写入通用寄存器，sel\_rf\_dst[0]表示该指令要将计算结果写入 rd 通用寄存器，sel\_rf\_dst[1]表示该指令要将计算结果写入 rt 通用寄存器,sel\_rf\_dst[2]表示该指令要将计算结果写入 31 号通用寄存器。rf\_waddr 表示要该条指令的计算结果要写入的通用寄存器的地址，data\_ram\_en表示该条指令是否要与内存中取值或者写入值，如果该条指令要从内存中取值或者写入值，那么它将被赋值为 1’b1,data\_ram\_wen 为四位宽，表示该条指令是否要写入寄存器，如果该条指令要将计算结果的第几个字节写入寄存器，那么对应位置的值设为 1。

跳转指令：先用br\_e表示这条指令是否为跳转指令，用rs\_ge\_z表示是否满足 rdata1的值大于等于 0，用rs\_le\_z表示是否满足rdata1的值小于等于0，用rs\_lt\_z表示是否满足 rdata1的值小于 0，rs\_eq\_rt表示是否满足rdata1是否等于radta2的值。br\_addr 表示跳转后的地址，根据不同的指令对地址做不同的计算，并将结果赋给br\_addr。

**第三部分 判断操作数来源**

用sel\_alu\_src1和sel\_alu\_src2来判断操作数来源，第一个操作数有三种来源，第二个操作数有四种来源，通过区分不同的指令进而分辨不同的操作数的来源。

传值用ID段得到的数据，分别给id\_to\_ex\_bus和br\_bus赋值，其中br\_bus是传给IF段的用于传输跳转指令的判断信号和跳转的地址。

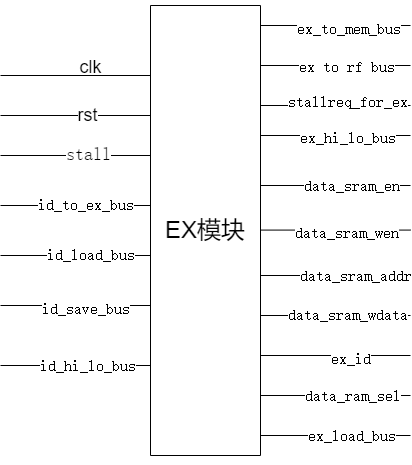
regfile模块接口如图所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | clk  k | 1 | 输入 | 时钟信号 |
| 2 | we | 1 | 输入 | 寄存器的写使能信号 |
| 3 | raddr1 | 5 | 输入 | 读取的第一个数的地址 |
| 4 | raddr2 | 5 | 输入 | 读取的第二个数的地址 |
| 5 | rdata1 | 32 | 输出 | 读取的第一个数的值 |
| 6 | rdata2 | 32 | 输出 | 读取的第二个数的值 |
| 7 | waddr | 5 | 输入 | 写入的地址 |
| 8 | wdata | 32 | 输入 | 写入寄存器的值 |

regfile模块说明：

regfile模块的作用是确定 rs 寄存器以及 rt 寄存器的值，判断 raddr1 是否为零，如果为零，就 把 32’b0 赋值给 rdata1,如果不为零，就把 raddr1 对应的寄存器的值赋值给 rdata1; 判断 raddr2 是否为零，如果为零，就把 32’b0 赋值给 rdata2,如果 不为零，就把 raddr2 对应的寄存器的值赋值rdata2。

表 3 regfile模块输入输出

**2.3 EX模块**

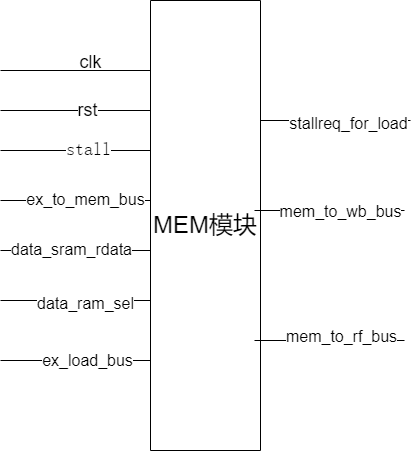
**整体说明：**

执行运算、计算地址和计算ALU结果。从ID/EX流水线寄存器中读取由寄存器1传来的值和寄存器2传来的值（或寄存器1传来的值和符号扩展过后的立即数的值），并用ALU将它们相加，结果值存入EX/MEM流水线寄存器。其中ALU模块已经提供，基本通过给alu提供控制信号就可以完成逻辑和算术运算，对于需要访存的指令在此段发出访存请求。接口如右图所示。

表 4 EX模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | id\_load\_bus | 5 | 输入 | ID段传递读的数据 |
| 6 | id\_save\_bus | 3 | 输入 | ID段传递写的数据 |
| 7 | ex\_to\_mem\_bus | 80 | 输出 | EX 段传给 MEM 段的数据 |
| 8 | ex\_to\_rf\_bus | 38 | 输出 | EX 段传给regfile段的数据 |
| 9 | id\_hi\_lo\_bus | 72 | 输入 | ID 段传给hilo段的数据 |
| 10 | ex\_hi\_lo\_bus | 66 | 输出 | EX 段传给hilo段的数据 |
| 11 | stallreq\_for\_ex | 1 | 输出 | 对EX段的stall请求 |
| 12 | data\_sram\_en | 1 | 输出 | 内存数据的读写使能信号 |
| 13 | data\_sram\_wen | 4 | 输出 | 内存数据的写使能信号 |
| 14 | data\_sram\_addr | 32 | 输出 | 内存数据存放的地址 |
| 15 | data\_sram\_wdata | 32 | 输出 | 要写入内存的数据 |
| 16 | ex\_id | 38 | 输出 | EX 段传给 ID 段的数据 |
| 17 | data\_ram\_sel | 4 | 输出 | 内存数据的选择信号 |
| 18 | ex\_load\_bus | 5 | 输出 | EX 段读取的数据 |

**功能说明：**

MEM模块，该模块有18个输入输出端口，包括clk，rst，stall，id\_to\_ex\_bus，id\_load\_bus，id\_save\_bus，id\_hi\_lo\_bus等7个输入端口和ex\_to\_mem\_bus，ex\_to\_rf\_bus，ex\_hi\_lo\_bus，stallreq\_for\_ex，data\_sram\_en，data\_sram\_wen，data\_sram\_addr，data\_sram\_wdata，ex\_id，data\_ram\_sel，ex\_load\_bus等11个输出端口，该模块还包含乘除法的部分实现。

**2.4 MEM模块**

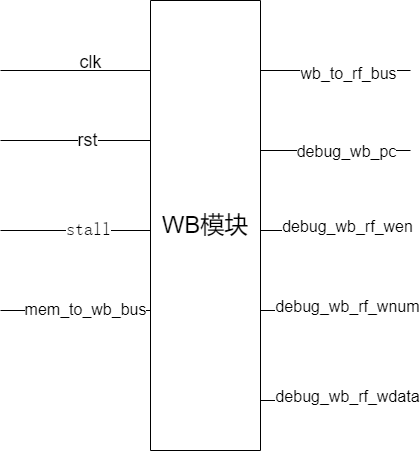
**整体说明：**

执行访问内存操作，从EX/MEM流水线寄存器中得到地址读取数据寄存器，并将数据存入MEM/WB流水线寄存器。接收并处理访存的结果，并选择写回结果对于需要访存的指令在此段接收访存结果。接口如上图所示。

表 5 MEM模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | data\_ram\_sel | 4 | 输入 | 内存数据的选择信号 |
| 7 | ex\_load\_bus | 5 | 输入 | EX 段读取的数据 |
| 8 | stallreq\_for\_load | 1 | 输出 | 对EX段的stall请求 |
| 9 | mem\_to\_wb\_bus | 70 | 输出 | MEM 传给 WB 段的数据 |
| 10 | mem\_to\_rf\_bus | 38 | 输出 | MEM段传给regfile段的数据 |

**功能说明：**

MEM模块，该模块有10个输入输出端口，包括clk、rst、stall、ex\_to\_mem\_bus、ex\_load\_bus作为输入，并将mem\_to\_wb\_bus和mem\_to\_rf\_bus作为输出。它能够完成load和store指令的执行，处理lb、lbu、lh、lhu、lw，store指令sb、sh等指令。对于load指令，它会根据地址最低两位来确定字节选择，并将结果写入RF寄存器；而store指令则会根据地址最低两位来确定字节写使能，并将数据写入数据RAM。

**2.5 WB模块**

**整体说明：**

将结果写回寄存器,从MEM/WB流水线寄存器中读取数据并将它写回图中部的寄存器堆中。接口如右图所示。

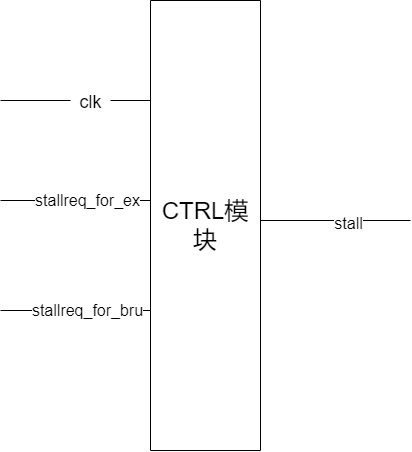
表 6 WB模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | debug\_wb\_pc | 32 | 输出 | 用来 debug 的 pc 值 |
| 7 | debug\_wb\_rf\_wen | 4 | 输出 | 用来 debug 的写使能信号 |
| 8 | debug\_wb\_rf\_wnum | 5 | 输出 | 用来 debug 的写寄存器地址 |
| 9 | debug\_wb\_rf\_wdata | 32 | 输出 | 用来 debug 的写寄存器数据 |

**功能说明：**

WB模块，该模块有9个输入输出端口，包括时钟（clk），复位（rst），stall，mem\_to\_wb\_bus，以及调试信号。输出包括wb\_to\_rf\_bus，以及debug用的调试信号。

该模块的主要功能是从MEM/WB流水线寄存器中读取数据，并将它写回图中部的寄存器堆中。它首先定义了一个名为mem\_to\_wb\_bus\_r的寄存器，用于存储从MEM/WB流水线寄存器中读取的数据，然后将数据写回寄存器堆中。最后，它将写回的数据通过wb\_to\_rf\_bus输出，以便在图中部的寄存器堆中更新数据。

**2.6 CTRL模块**

**整体说明：**

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。接口如右图所示。

表 7 CTRL模块输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | stallreq\_for\_ex | 1 | 输入 | 执行阶段的指令是否请求流水线暂停 |
| 3 | stallreq\_for\_bru | 5 | 输入 | Load命令是否请求流水线暂停 |
| 4 | stall | 6 | 输出 | 暂停信号 |

**功能说明：**

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 阶段及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。stall[0]为 1 表示没有暂停，1-5为 1时分别 代表if 段、id 段、ex 段、 mem 段、wb 段暂停。

**2.7 HILO寄存器模块**

**整体说明：**

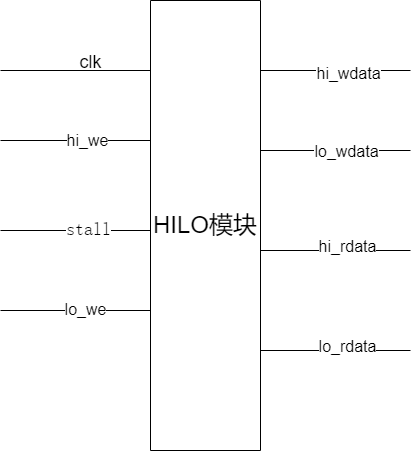
hi和lo属于协处理器，不在通用寄存器的范围内，这两个寄存器主要是在用来处理乘法和除法。以乘法作为示例，如果两个整数相乘，那么乘法的结果低位保存在lo寄存器，高位保存在hi寄存器。 当然，这两个寄存器也可以独立进行读取和写入。读的时候，使用mfhi、mflo；写入的时候，用mthi、mtlo。和通用寄存器不同，mfhi、mflo是在执行阶段才开始从hi、lo寄存器获取数值的。写入则和通用寄存器一样，也是在写回的时候完成的。接口如右图所示。

表 8 HILO寄存器输入输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名 | 宽度 | 输入/输出 | 作用 |
| 1 | clk | 1 | 输入 | 时钟信号 |
| 2 | stall | 6 | 输入 | 控制暂停信号 |
| 3 | hi\_we | 1 | 输入 | hi寄存器的写使能信号 |
| 4 | lo\_we | 1 | 输入 | lo寄存器的写使能信号 |
| 5 | hi\_wdata | 32 | 输出 | Hi寄存器写的数据 |
| 6 | lo\_wdata | 32 | 输出 | Lo寄存器写的数据 |
| 7 | hi\_rdata | 32 | 输出 | Hi寄存器读的数据 |
| 8 | lo\_rdata | 32 | 输出 | Lo寄存器读的数据 |

**功能说明：**

当hi\_we和lo\_we均为1时，寄存器reg\_hi和reg\_lo同时将hi\_wdata和lo\_wdata写入。当hi\_we为0，lo\_we为1时，reg\_lo将lo\_wdata写入；当hi\_we为1，lo\_we为0时，reg\_hi将hi\_wdata写入。hi\_rdata和lo\_rdata分别输出reg\_hi和reg\_lo中的数据。

1. **实验感受及建议**

**3.1 王烁天部分**

在实验过程中，我对 GitHub 的使用愈发熟练，能够借助其搭建仓库、审阅代码并进行版本管理，为团队协作带来了显著效率提升。调试时，我们会在波形图中针对可疑的信号进行重点关注，尤其结合提示的 PC 值及其附近相关信号，从而快速定位并分析问题根源。

**3.2 田人溥部分**

在本次实验中，我对流水线的整体运行过程有了更深入的认识，将课堂所学的原理真正运用于实践。尽管我在项目中负责的部分可能不算多，但要想顺利完成那些指令的插入和功能实现，必须先搞懂所有代码的整体运行逻辑，真正理解流水线各个阶段的具体运行细节。因此，我也花费了不少时间上网查阅资料、解决疑惑。

通过这次实验，我也进一步体会到团队协作的重要性。要想让各项任务高效完成，必须做好明确的分工并且与队友保持充分的沟通，这样才能最大限度地发挥团队的力量。

总之，这次实验不仅让我学到了一种新的编程方法，更加深入地探究了流水线的运行模式与细节，也让我真切感受到团队合作的价值和意义。

**3.3 郑一夫部分**

首先，在实验内容方面，这次使用了全新的平台与实验模式，考核方式也别具一格，让我掌握了不少全新的技能与方法。例如，利用 GitHub 协作大幅提升了小组间代码同步与版本管理的效率；CG 实验平台的使用则让我们免去了繁琐的软件安装与调试；在调试过程中，更是对 CPU 的五级流水线有了更为深入的认识与实践。

其次，在小组合作上，这次实验让我对团队协作有了新的体会。我们小组在任务分配上非常明确，队友之间相互帮助、共同进步，这让我深切感受到高效团队的重要价值，同时也刷新了我对“最高效团队合作”的认知。

总之，这是一段令人愉悦的实验经历。我们不仅从新颖的实验方式和老师、助教的指导中受益良多，也通过团队合作的过程收获了更多乐趣与成就感。

**4. 参考资料**

1、张晨曦 著《计算机体系结构》（第二版） 高等教育出版社

2、雷思磊 著《自己动手写 CPU》 电子工业出版社

3、（美）DavidA.Patterson、John L.Hennessy 著 《计算机组 成与设计：硬件、软件接口（原书第 4 版）》

4、Yale N.Patt 著 《计算机系统概论（原书第 2 版）》

5、龙芯杯官方的参考文档