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未来实验班

组长：孙毅 20226483

组员：张硕昌 20226514

报告日期：2025.1.3

目录

[1. 实验概括 3](#_Toc8987)

[1.1 工作量 3](#_Toc27081)

[1.2 总体设计 3](#_Toc31182)

[1.2.1 取指 （IF） 4](#_Toc515)

[1.2.2 指令译码 （ID） 5](#_Toc21319)

[1.2.3 执行 （EX） 5](#_Toc10693)

[1.2.4 访存（MEM） 6](#_Toc28566)

[1.2.5 写回 （WB） 6](#_Toc2483)

[1.2.6 实验实现： 7](#_Toc25183)

[1.3 MIPS 指令格式 7](#_Toc26177)

[1.4 不同流水段之间的连线图 9](#_Toc26654)

[1.5 指令完成度 10](#_Toc21613)

[1.6 程序运行环境及使用工具 10](#_Toc10882)

[2. 单个流水段说明 11](#_Toc17705)

[2.1 IF 段 11](#_Toc5293)

[2.1.1 整体功能说明 11](#_Toc9512)

[2.1.2 端口介绍 11](#_Toc7765)

[2.1.3 信号介绍 12](#_Toc31716)

[2.1.4 修改内容 12](#_Toc27936)

[2.2 ID 段 13](#_Toc3106)

[2.2.1 整体功能说明 13](#_Toc26032)

[2.2.2 端口介绍 13](#_Toc18743)

[2.2.3 信号介绍 14](#_Toc21050)

[2.2.4 regfile 功能模块介绍 16](#_Toc20446)

[2.2.5 修改内容 17](#_Toc12952)

[2.3 EX 段 17](#_Toc18022)

[2.3.1 整体功能说明 17](#_Toc10812)

[2.3.2 端口介绍 18](#_Toc26477)

[2.3.3 信号介绍 18](#_Toc25198)

[2.3.4 修改内容 19](#_Toc21404)

[2.4 MEM 段 20](#_Toc11189)

[2.4.1 整体功能介绍 20](#_Toc27118)

[2.4.2 端口介绍 20](#_Toc16229)

[2.4.3 信号介绍 21](#_Toc9100)

[2.4.4 修改内容 22](#_Toc19100)

[2.5 WB 段 22](#_Toc5969)

[2.5.1 整体功能介绍 22](#_Toc30266)

[2.5.2 端口介绍 23](#_Toc10841)

[2.5.3 信号介绍 23](#_Toc31644)

[2.5.4 修改内容 24](#_Toc24160)

[3. 实验感受及改进意见 24](#_Toc5151)

[3.1 孙毅 24](#_Toc29619)

[3.2 张硕昌 25](#_Toc16815)

[4. 参考资料 25](#_Toc1023)

# 1. 实验概括

## 1.1 工作量

工作量由高到低：孙毅、张硕昌

## 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，以提高指令的执行效率和吞吐量。这种五级流水线的设计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。

在这里，结合MIPS处理器的特点，我们整体的处理分为五部分，分别为IF（取指）、ID（指令译码）、EX（执行）、MEM（访存）以及WB（写回）五级，在这里将指令执行分为五个阶段，保证每个阶段的指令可以并行进行，提高效率。

值得注意的是，每条指令的执行都需要五个时钟周期，在对应时钟周期的上升沿来临时，该指令所代表的一系列数据和控制信息会转移到下一级处理。例如，可能在第一个时钟周期取指令，在第二个周期开始译码，在第三个周期进行运算，依此类推，直到指令执行完毕。

具体的DLX 的基本流水线示意图如下所示（来自PPT）：

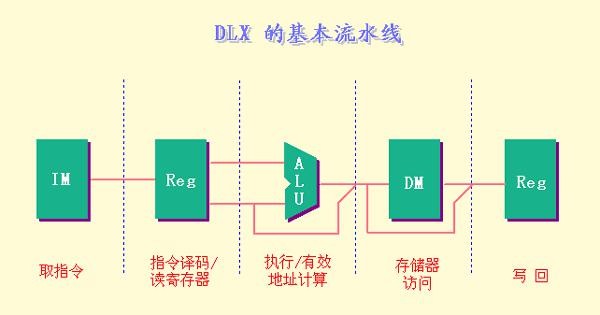


图 1.1 DLX 的流水线

之后我们详细了解这五个周期。

### 1.2.1 取指 （IF）

在这里我们每一条指令都是占用4字节，提前说明

IR ← Mem[PC]

NPC ← PC＋4

该周期的功能是从指令存储器中取出指令。

这里的具体操作如下：

首先从PC寄存器中获取指令地址（PC），然后再指令寄存器MEM中取出对应的指令信息，并存到指令寄存器IR中，之后进行下一条指令地址的计算（PC+4）并存储到NPC寄存器，之后IR和NPC存入IF-ID的流水段寄存器便于ID阶段的使用

### 1.2.2 指令译码 （ID）

A <- Regs[IR6..10] (Regs[rs])

B <- Regs[IR11..15] (Regs[rt])

Imm <- (IR16)16 ## IR16..31

该阶段是指令解码部分，具体是对于R型指令（加减法等）以及I型指令（立即数等）中的寄存器和常数的提取过程

在这个阶段我们将指令寄存器IR的6-10位取出，从寄存器中取出与之对应的寄存器内容存到A，这里取出的是源寄存器的值，，对于B，我们取出的是目标寄存器的值，而Imm的立即数操作是提取指令的立即数部分来进行符号拓展，具体的拓展方式我们一共有四种，如下：逻辑拓展、算术拓展、为LUI指令进行的拓展以及为分支指令进行的拓展。

### 1.2.3 执行 （EX）

存储器访问：

这部分主要针对的是LOAD以及STORE指令，对于这部分，ALU通常需要计算访问内存的地址

ALIOutput <- A + Imm

寄存器-寄存器ALU操作：

这里是对于寄存器之间的操作，通常是从rs和rt提取两个值A和B，并进行需要的ALU运算，最后将结果存储到ALUOutput中

ALUOutput <- A op B

寄存器-立即数ALU操作：

同寄存器-寄存器的ALU，不同的是rt寄存器的数值变为Imm立即数

ALUOutput <- A op Imm

分支操作：

对于分支对应指令，我们需要计算跳转的目标地址，这里NPC是下一个PC值（PC + 4），Imm是立即数（偏移量）

ALUOutput <- NPC + Imm

这部分是计算分支指令条件，判断是否满足跳转条件的

Cond <- A op 0

这个阶段（EX），CPU根据不同类型的指令执行对应的运算，每种指令会计算出不同的结果，并通过控制信号来知道数据通路的选择。

同时，对于LW和SW指令，在这个周期需要进行内存的访问，对于分支指令需要计算跳转目标PC以及判断是否进行分支

### 1.2.4 访存（MEM）

在这个周期，我们处理的DLX指令只有Load、Store以及分支指令

存储器访问：

LMD <- Mem[ALUOutput] 或者 Mem[ALUOutput] <- B

前半部分通常是LW指令，将从内存读取到的数据存放在LMD寄存器

后面部分通常是SW指令，将B中的数据存放到内存中的指定地址

分支操作：

If(cond) PC <- ALUOutput

这里处理分支指令，如果cond为真，也就是满足跳转条件，我们就更新PC为ALUOutput（这个地址是分支目标地址-NPC + Imm）

Else PC <- NPC

如果为假，则不进行跳转，下一个地址仍然是NPC

在MEM段，只有内存访问指令（LW和SW）执行内存读写操作，除此之外的指令都不涉及内存操作

### 1.2.5 写回 （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

这类执行寄存器之间的算数逻辑运算，在该阶段指令的执行结果（ALUOutput）会被写回到目标寄存器（rd）

寄存器-立即数型ALU指令

Regs[IR11..15] (rt) <- ALUOutput

该指令处理寄存器与立即数之间的算术逻辑运算，将ALU运算结果（ALUOutput）写回到寄存器（rt）

Load指令

Regs[IR11..15] (rt) <- LMD

这条指令执行的是从内存加载数据到寄存器，也就是从内存（LMD）读取数据写回到寄存器（rt）

这里我们说一下时钟信号问题：

在 流水线 CPU 设计中，所有寄存器都会根据时钟信号的上升沿或下降沿进行操作。一般来说，寄存器的 读取操作 是在 上升沿 进行的，而 写入操作 是在 下降沿 进行的。

由于 CPU 中的 流水线寄存器 和 时钟信号的周期性，这可能会导致 写回寄存器的时间与读取寄存器的时间 在某些情况下重叠。如果此时出现读取和写入同一个寄存器的情况，就会发生 结构冲突（Structural Hazard），即同一个寄存器的读操作和写操作发生冲突。

所以在这里使用了一种数据转发机制

### 1.2.6 实验实现：

分支指令：4个时钟周期（转到ID段需2个）

存储指令（store）：4个时钟周期

其他指令：5个时钟周期

## 1.3 MIPS 指令格式

在实验中我们的操作是建立在MIPS32指令系统上完成的

所有指令长度均为32比特。

除个别指令外，所有指令的格式均为立即数型（I-Type）、跳转型（J-Type）和寄存器型（R-Type）三种类型中的一种。三类指令格式如下。

1. I-type

该类指令包括load和store指令、里技术质量、分支指令和部分跳转指令

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| 操作码 | Rs | Rt | 立即数（Imm） |
| 6 | 5 | 5 | 16 |

具体如下：

load 指令

访存有效地址：Regs[rs]＋immediate

从存储器取来的数据放入寄存器 rt

store 指令

访存有效地址：Regs[rs]＋immediate

要存入存储器的数据放在寄存器 rt 中

立即数指令

Regs[rt] ← Regs[rs] op immediate

分支指令

转移目标地址：Regs[rs]＋immediate，rt

转移目标地址为 Regs[rs]

1. J-type

该类包括跳转指令等、

|  |  |
| --- | --- |
| 6 | 6 |
| 操作码 | 与PC相加的偏移量 |

31 26 25 0

1. R-type

该类包括ALU指令、专用寄存器读写指令，move指令等

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 5 | 5 | 5 | 5 | 6 |
| 操作码 | Rs | Rt | Rd | Sa | Function |

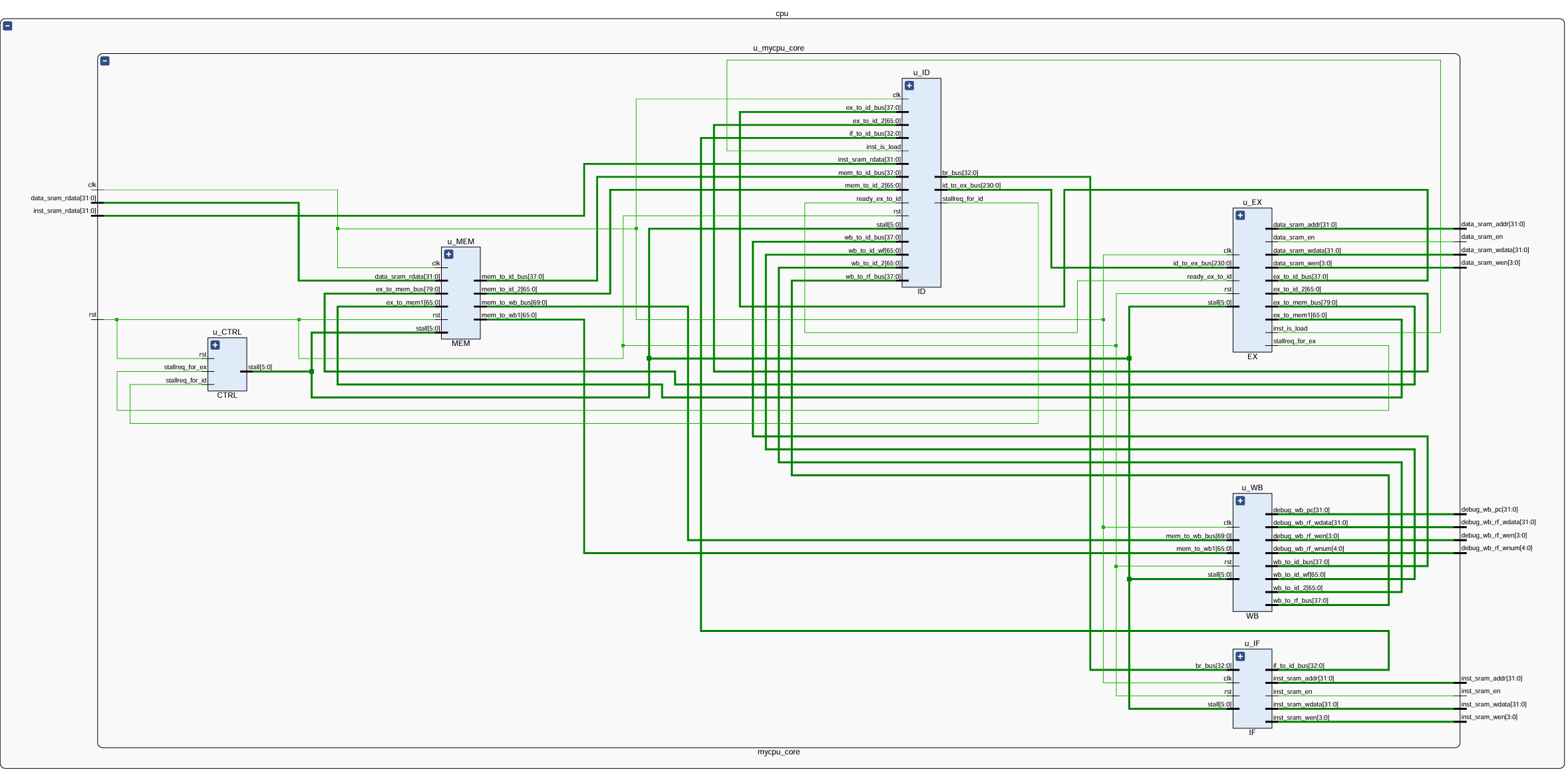
31 26 25 21 20 16 15 11 10 6 5 0

处理器需要实现的指令包括除4条非对齐指令外的所有MIPS I指令以及MIPS32中的ERET指令，有14条算

术运算指令、8条逻辑运算指令，6条移位指令、8条分支跳转指令、4条数据移动指令、2条自陷指令、12条访存

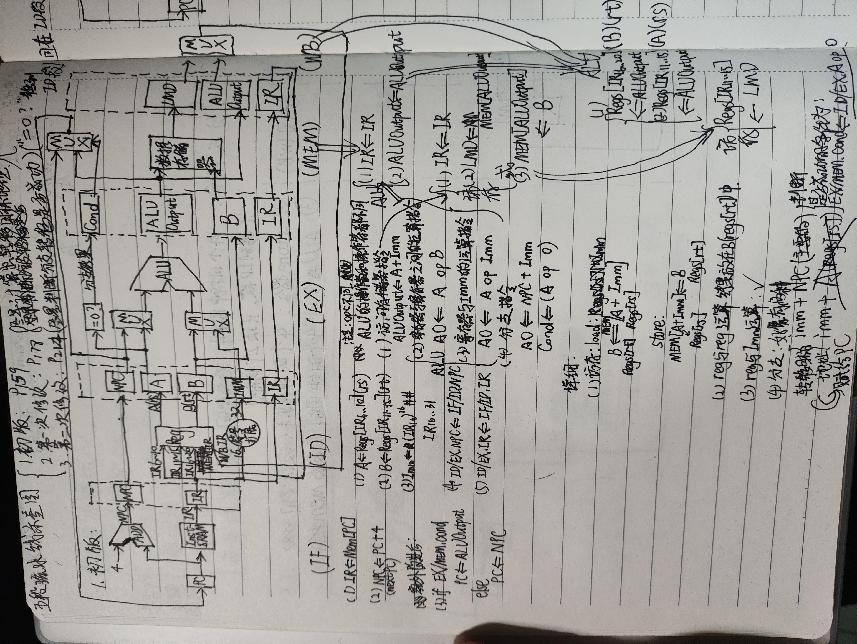
指令、3条特权指令，共计57条。

## 1.4 不同流水段之间的连线图

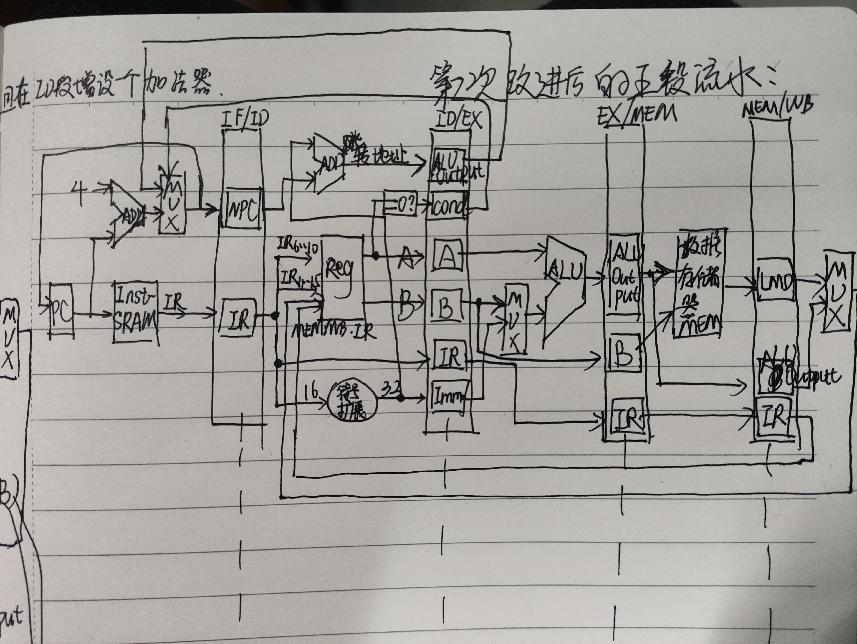


图表 1CPU结构图

以下是我们组手画的五段流水线及其改进两次之后的五段流水：



原版五段流水线



改进两次后的五段流水线

## 1.5 指令完成度

目前添加了逻辑运算指令、算术运算指令、移位指令、分支跳转指令、数据移动质量、访存指令等49条

具体如下：

wire 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;

## 1.6 程序运行环境及使用工具

编程软件：vivado 2019.2

编译语言：verilog

PC环境：windows 11 64位

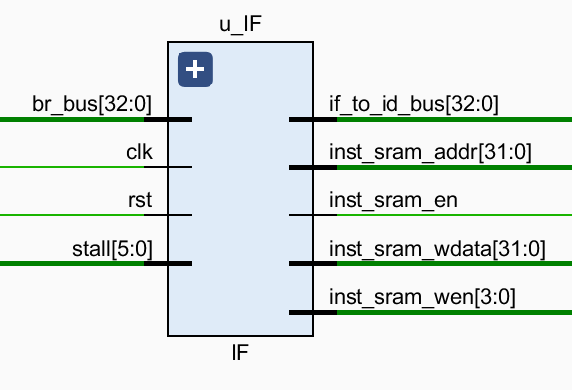
# 2. 单个流水段说明

## 2.1 IF 段

### 2.1.1 整体功能说明

IF（Instruction Fetch）段是CPU流水线的第一个阶段，负责从指令存储器中取出指令。该模块根据程序计数器（PC）的值生成指令存储器的地址，并在时钟上升沿更新PC值。如果分支信号有效，PC会跳转到分支目标地址；否则，PC会顺序递增。IF段还负责控制指令存储器的使能信号，确保在复位或流水线暂停时停止取指操作。最终，IF段将取出的指令地址和使能信号传递给下一阶段（ID段）。

### 2.1.2 端口介绍



（1）输入端口：br\_bus[32:0],clk,rst,stall[5:0]

（2）输出端口：if\_to\_id\_bus[32:0], inst\_sram\_addr[31:0], inst\_sram\_en, inst\_sram\_wdata[31:0], inst\_sram\_wen[3:0]

### 2.1.3 信号介绍

（1）clk：时钟信号，驱动模块的时序逻辑。所有寄存器的更新（如pc\_reg和ce\_reg）都在时钟的上升沿触发。

（2）rst：复位信号，高电平有效。当rst为高时，pc\_reg被初始化为32'hbfbf\_fffc，ce\_reg被初始化为1'b0，表示复位状态下停止取指操作。

（3）stall[StallBus-1:0]：流水线暂停信号。stall[0]用于控制PC和指令存储器使能信号的更新。当stall[0] == NoStop时，PC和ce\_reg正常更新；否则，暂停更新。剩余五位分别为if，id，ex，mem，wb对应的暂停信号。

（4）br\_bus[BR\_WD-1:0]：分支信号总线，包含分支有效信号br\_e和分支目标地址br\_addr。通过assign {br\_e, br\_addr} = br\_bus解码得到。

（5）if\_to\_id\_bus[IF\_TO\_ID\_WD-1:0]：输出到ID 阶段的信号总线，包含指令存储器使能信号ce\_reg和当前 PC 值pc\_reg。通过assign if\_to\_id\_bus = {ce\_reg, pc\_reg}赋值。

（6）inst\_sram\_en：指令存储器使能信号，直接连接到ce\_reg。当ce\_reg为高时，指令存储器可以响应读请求。

（7）inst\_sram\_wen[3:0]：指令存储器写使能信号，固定为 4'b0，表示指令存储器只读不写。

（8）inst\_sram\_addr[31:0]：指令存储器地址，直接连接到pc\_reg。表示当前要读取的指令地址。

（9）inst\_sram\_wdata[31:0]：写入指令存储器的数据，固定为 32'b0，表示不写数据。

（10）pc\_reg[31:0]：程序计数器寄存器，存储当前指令地址。在时钟上升沿更新，如果rst为高，pc\_reg被初始化为32'hbfbf\_fffc。如果stall[0] == NoStop，pc\_reg更新为next\_pc。

（11）ce\_reg：指令存储器使能寄存器，控制指令存储器的使能状态。在时钟上升沿更新，如果rst为高，ce\_reg被初始化为1'b0。如果stall[0] == NoStop，ce\_reg更新为1'b1。

（12）next\_pc[31:0]：下一条指令地址，如果分支有效（br\_e为高），next\_pc为分支目标地址br\_addr。否则，next\_pc为当前 PC 值加 4（pc\_reg + 32'h4）。

（13）br\_e：分支有效信号，指示是否需要跳转。从br\_bus中解码得到。

（14）br\_addr[31:0]：分支目标地址，当分支有效时，PC跳转到该地址。从br\_bus中解码得到。

### 2.1.4 修改内容

没有修改任何内容。

## 2.2 ID 段

### 2.2.1 整体功能说明

ID（Instruction Decode）段是CPU流水线的第二阶段，负责对从IF段取出的指令进行译码，并生成控制信号和数据信号。ID段从IF段接收指令数据，从WB段接收写回数据，并将译码结果传递到EX段。其主要功能包括：

（1）指令译码：解析指令的操作码和功能码，生成控制信号（如ALU操作、寄存器写使能等）。

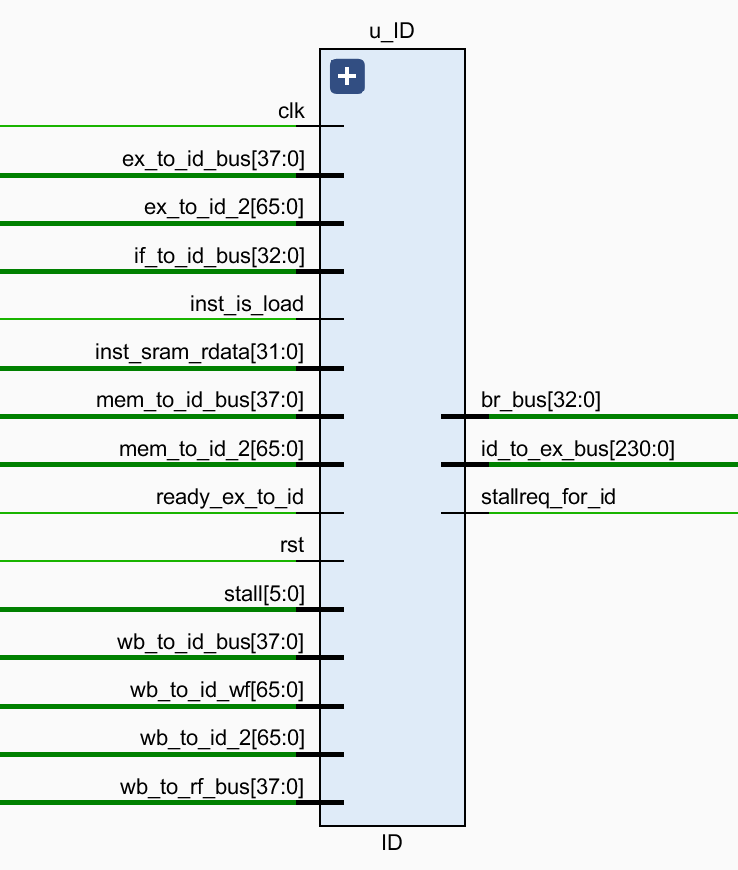
（2）寄存器读取：从寄存器文件中读取源操作数（rs和rt的值）。

（3）分支判断：根据指令类型和操作数判断是否满足分支条件，生成分支信号。

（4）流水线控制：生成流水线暂停请求信号（stallreq），处理数据冒险和结构冒险。

（5）高低位寄存器操作：支持对高低位寄存器的读写操作。

### 2.2.2 端口介绍



（1）输入端口： clk, rst, stall[5:0], ex\_to\_id\_bus[37:0], mem\_to\_id\_bus[37:0],wb\_to\_id\_bus[37: 0], ex\_to\_id\_2[65:0], mem\_to\_id\_2[65:0], wb\_to\_id\_2[65:0], if\_to\_id\_bus[32:0], inst\_sram\_rdata[31:0], inst\_is\_load,wb\_to\_rf\_bus[37:0], wb\_to\_id\_wf[65:0], ready\_ex\_to\_id

（2）输出端口：br\_bus[32:0],id\_to\_ex\_bus[230:0],stallreg\_for\_id

### 2.2.3 信号介绍

输入输出信号：

（1）clk：时钟信号，驱动模块的时序逻辑。所有寄存器的更新（if\_to\_id\_bus\_r）都在时钟的上升沿触发。

（2）rst：复位信号，高电平有效。当rst为高时，if\_to\_id\_bus\_r被初始化为全0，表示复位状态下停止译码操作。

（3）stall[StallBus-1:0]：流水线暂停信号。stall[1]用于控制ID段的译码操作。如果stall[1] == Stop，ID 段暂停译码；否则，正常译码。

（4）stallreq：流水线暂停请求信号。当ID段检测到数据冒险（加载指令后立即使用）时，stallreq为高，请求流水线暂停。

（5）stallreq\_for\_id：ID 段的暂停请求信号。当检测到加载指令（inst\_is\_load）且目标寄存器与后续指令的源寄存器冲突时，stallreq\_for\_id为高。

（6）ex\_to\_id\_bus[37:0]：从EX段传递到ID段的总线信号，包含EX段的执行结果和状态信息。

（7）mem\_to\_id\_bus[37:0]：从MEM段传递到ID段的总线信号，包含MEM段的执行结果和状态信息。

（8）wb\_to\_id\_bus[37:0]：从WB段传递到ID段的总线信号，包含WB段的写回数据和状态信息。

（9）if\_to\_id\_bus[IF\_TO\_ID\_WD-1:0]：从IF段传递到ID段的总线信号，包含当前指令的地址（id\_pc）和指令存储器使能信号（ce）。

（10）inst\_sram\_rdata[31:0]：从指令SRAM读取的指令数据，即当前需要译码的指令。

（11）wb\_to\_rf\_bus[WB\_TO\_RF\_WD-1:0]：从WB段传递到寄存器文件的总线信号，包含写使能信号（wb\_rf\_we）、写地址（wb\_rf\_waddr）和写数据（wb\_rf\_wdata）。

（12）id\_to\_ex\_bus[ID\_TO\_EX\_WD-1:0]：从ID段传递到EX段的总线信号，包含译码后的控制信号和数据信号（如ALU操作码、操作数选择信号、寄存器写地址等）。

（13）br\_bus[BR\_WD-1:0]：分支信号总线，包含分支使能信号（br\_e）和分支目标地址（br\_addr）。

（14）inst\_is\_load：指示当前指令是否为加载指令（比如lw、lb）。用于检测数据冒险。

（15）ready\_ex\_to\_id：EX段到ID段的就绪信号，用于控制流水线暂停逻辑。

1. wb\_to\_id\_wf[65:0]：从WB段传递到ID段的高低寄存器写信号，包含高低位寄存器的写使能信号和写数据。

ID内部信号（解包信息）：

（1）if\_to\_id\_bus\_r[IF\_TO\_ID\_WD-1:0]：存储从IF段传递到ID段的总线信号，用于在ID阶段使用。在时钟上升沿更新。如果rst为高，初始化为全0；如果stall[1]==Stop，清零；否则，更新为if\_to\_id\_bus。

（2）inst[31:0]：当前需要译码的指令。如果inst\_stall\_en1为高，使用inst\_stall1（暂停时的指令）；否则，使用inst\_sram\_rdata。

（3）id\_pc[31:0]：当前指令的程序计数器（PC）值。从if\_to\_id\_bus\_r中提取。

（4）ce：指令SRAM的使能信号。从if\_to\_id\_bus\_r中提取。

（5）wb\_rf\_we：WB段对寄存器文件的写使能信号，从wb\_to\_rf\_bus中提取。

（6）wb\_rf\_waddr[4:0]：WB段对寄存器文件的写地址，从wb\_to\_rf\_bus中提取。

（7）wb\_rf\_wdata[31:0]：WB段对寄存器文件的写数据从wb\_to\_rf\_bus中提取。

（8）opcode[5:0]：指令的操作码，从inst[31:26]中提取。

（9）rs[4:0], rt[4:0], rd[4:0], sa[4:0]：源寄存器 1、源寄存器 2、目标寄存器和移位量。分别从inst[25:21]、inst[20:16]、inst[15:11]和inst[10:6]中提取。

（10）func[5:0]：指令的功能码。从inst[5:0]中提取。

（11）imm[15:0]：指令中的立即数。从inst[15:0]中提取。

（12）instr\_index[25:0]：跳转指令中的跳转地址。从inst[25:0]中提取。

（13）code[19:0]：指令中的特殊字段。从inst[25:6]中提取。

（14）base[4:0]：基址寄存器地址。从inst[25:21]中提取。

（15）offset[15:0]：偏移量。从inst[15:0]中提取。

（16）sel[2:0]：选择信号。从inst[2:0]中提取。

（17）op\_d[63:0], func\_d[63:0]：操作码和功能码的解码结果。通过decoder\_6\_64模块将opcode和func解码为独热码。

（18）rs\_d[31:0], rt\_d[31:0], rd\_d[31:0], sa\_d[31:0]：寄存器值和移位量的解码结果。通过decoder\_5\_32模块将rs、rt、rd和sa解码为独热码。

（19）sel\_alu\_src1[2:0], sel\_alu\_src2[3:0]：选择ALU操作数的来源。

根据指令类型生成选择信号。例如，sel\_alu\_src1[0]为1表示 ALU 的第一个操作数来自rs。

（20）alu\_op[11:0]：ALU 操作的控制信号。根据指令类型生成ALU操作类型（如加法、减法、逻辑与等）。

（21）data\_ram\_en：数据存储器的使能信号。如果当前指令是加载或存储指令（如lw、sw），data\_ram\_en为高。

（22）data\_ram\_wen[3:0]：数据存储器的写使能信号。如果当前指令是存储指令（如sw），data\_ram\_wen为4'b1111；否则为4'b0000。

（23）rf\_we：寄存器文件的写使能信号。如果当前指令需要写回寄存器（如add、lw），rf\_we为高。

（24）rf\_waddr[4:0]：寄存器文件的写地址。根据sel\_rf\_dst选择写地址来源（rd、rt或31）。

（25）sel\_rf\_res：选择寄存器文件写数据的来源。如果当前指令是加载指令（如lw），sel\_rf\_res为1，表示写数据来自加载结果；否则为0，表示写数据来自ALU结果。

（26）rdata1[31:0], rdata2[31:0]：从寄存器文件读取的数据，分别对应rs和rt的值。通过regfile模块读取寄存器文件中的数据。

（27）br\_e：分支使能信号。根据指令类型和操作数判断是否满足分支条件。例如，inst\_beq && rs\_eq\_rt表示分支相等指令且rs和rt的值相等。

（28）br\_addr[31:0]：分支目标地址。根据指令类型计算分支目标地址。例如，inst\_beq的分支目标地址为pc\_plus\_4 + {{14{inst[15]}}, inst[15:0], 2'b0}。

（29）pc\_plus\_4[31:0]：当前指令地址加4，即下一条指令的地址。即id\_pc + 32'h4。

（30）rs\_eq\_rt：表示rs和rt的值是否相等。即rdata1==rdata2。

（31）lo\_hi\_r[1:0], lo\_hi\_w[1:0]：高低位寄存器的读写使能信号。根据指令类型生成高低位寄存器的读写信号。例如，inst\_mflo时，lo\_hi\_r[0]为高。

（32）hi\_o[31:0], lo\_o[31:0]：从高低位寄存器读取的数据。通过regfile模块读取高低位寄存器中的数据。

（33）data\_ram\_read[3:0]：数据存储器的读控制信号。根据指令类型生成读控制信号。例如，inst\_lw时，data\_ram\_read为4'b1111。

### 2.2.4 regfile 功能模块介绍

Regfile（寄存器文件）是CPU中用于存储和读取数据的核心模块，包含32个通用寄存器以及高低位寄存器（HI和LO）。主要功能如下：

（1）数据存储：通过写使能信号（we）和写地址（waddr），将数据写入指定的寄存器。

（2）数据读取：通过读地址（raddr1和raddr2），从寄存器文件中读取数据。

（3）数据冒险处理：检查执行阶段（EX）、访存阶段（MEM）和写回阶段（WB）的写回数据，优先使用最新的数据，避免数据冒险。

（4）高低位寄存器操作：支持对高低位寄存器的读写操作，用于乘除法等指令。

（5）移位操作支持：通过inst\_lsa信号支持加载存储地址（LSA）指令的移位操作。

## 2.2.5 修改内容

（1）在ID段，对ex\_to\_id,mem\_to\_id,wb\_to\_id 进行数据通路连线，使用ex\_to\_id\_bus解决了第一个报错。

（2）在ID段，添加stallreq\_for\_id信号，并对应的在id和ex和mycpu\_core中添加了inst\_is\_load控制信号。

（3）在ID段，添加inst\_stall信号，并对应的在id和ex和mycpu\_core中添加了ready\_ex\_to\_id 控制信号。

（4）在ID段，添加对剩余60条指令（不包括已经写好的四条指令）的识别信号，及其对应的控制信号和数据信号，以及这些信号的打包传递。

（5）在ID段，添加data\_ram\_read相关的控制信号，为lw，lb等访存指令（point43）的数据读取做数据通路。

（6）在ID段，添加与hi，lo寄存器相关定义和数据通路（hi lo应用在三种地方：mul，div，数据迁移）。

## 2.3 EX 段

### 2.3.1 整体功能说明

EX阶段是流水线中的执行阶段，主要完成算术逻辑运算、数据存储器访问和乘除法运算等任务。该模块接收来自ID阶段的指令信息，生成控制信号，进行指令的运算处理，并将结果传递到MEM阶段。具体功能包括：

（1）算术运算（ALU）：根据控制信号，选择ALU的输入数据源，执行算术、逻辑或移位操作，生成运算结果。

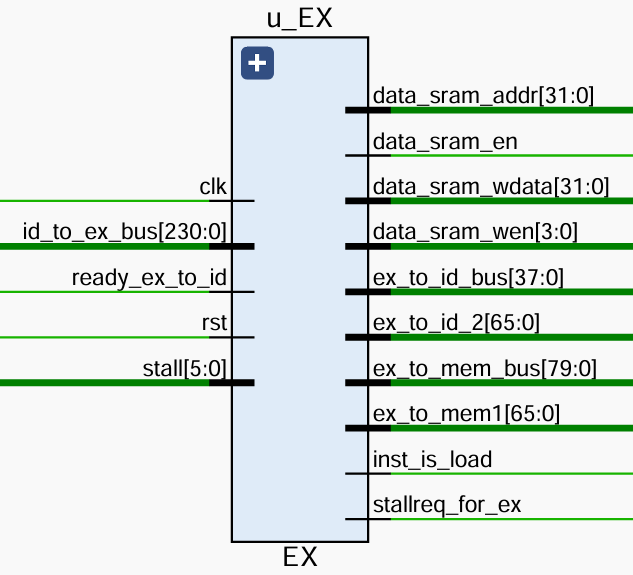
（2）数据存储器访问：根据指令类型（如Load/Store），生成数据存储器访问信号，读写数据。

（3）乘法与除法：支持乘法和除法操作，处理相应的运算，写回高低寄存器。

（4）寄存器文件更新：处理指令结果，更新寄存器文件（RF），并将相关信号传递给MEM阶段。

（5）流水线暂停与延迟槽：通过控制信号和STALL机制，决定是否暂停或继续执行。

### 2.3.2 端口介绍



（1）输入端口：

clk,rst,id\_to\_ex\_bus[230:0],stall[5:0]

（2）输出端口：

data\_sram\_addr[31:0], data\_sram\_en, data\_sram\_wdata[31:0],

data\_sram\_wen[3:0], ex\_to\_id\_bus[37:0]

ex\_to\_id2[65:0], ex\_to\_mem\_bus[79:0], ex\_to\_mem1[65:0],

inst\_is\_load, ready\_e x\_to\_id, stall\_for\_ex

### 2.3.3 信号介绍

1. id\_to\_ex\_bus：包含ID阶段传递到EX阶段的所有信息。该总线包括PC值、指令、ALU操作码、控制信号、寄存器数据等。

ex\_pc [32-bit]：当前指令的PC值，用于指令的地址计算。

inst [32-bit]：当前指令。

alu\_op [12-bit]：ALU操作码，控制ALU执行何种操作（如加法、减法、与、或等）。

sel\_alu\_src1 [3-bit]：选择ALU操作数1的来源（如寄存器、PC等）。

sel\_alu\_src2 [4-bit]：选择ALU操作数2的来源（如立即数、寄存器等）。

data\_ram\_en [1-bit]：数据存储器使能信号，指示是否访问数据存储器。

data\_ram\_wen [4-bit]：数据存储器写使能信号，指示要写入的数据字节。

rf\_we [1-bit]：寄存器文件写使能信号，指示是否写回寄存器文件。

rf\_waddr [5-bit]：写寄存器的地址。

sel\_rf\_res [1-bit]：选择ALU结果或数据存储器读取结果，用于寄存器文件写入。

rf\_rdata1 [32-bit]：寄存器文件读取的第一个数据。

rf\_rdata2 [32-bit]：寄存器文件读取的第二个数据。

1. ex\_to\_mem\_bus：传递给MEM阶段的总线信号，包含ALU结果、控制信号等。

ex\_pc [32-bit]：传递到MEM阶段的PC值。

data\_ram\_en [1-bit]：数据存储器使能信号。

data\_ram\_wen [4-bit]：数据存储器写使能信号。

sel\_rf\_res [1-bit]：决定寄存器写入的数据来源（ALU结果或存储器读取数据）。

rf\_we [1-bit]：寄存器文件写使能信号。

rf\_waddr [5-bit]：寄存器文件写地址。

ex\_result [32-bit]：ALU或其他计算结果。

data\_ram\_read [4-bit]：数据存储器读取字节。

1. ex\_to\_id\_bus：EX阶段的信息反馈到ID阶段，主要用于数据旁路和控制信号传递。

rf\_we [1-bit]：寄存器文件写使能信号。

rf\_waddr [5-bit]：写寄存器的地址。

ex\_result [32-bit]：EX阶段的结果，用于数据旁路。

4. data\_sram\_en [1-bit]：数据存储器使能信号，控制数据存储器是否被激活。

5. data\_sram\_wen [4-bit]：数据存储器写使能信号，控制要写入内存的字节。

6. data\_sram\_addr [32-bit]：数据存储器地址信号，指定要访问的内存位置。

7. data\_sram\_wdata [32-bit]：写入数据存储器的数据。

8. stallreq\_for\_ex [1-bit]：EX阶段请求流水线暂停信号，通常由除法器或乘法器引发，用于防止数据冒险。

9. inst\_is\_load [1-bit]：指示当前指令是否为Load指令，用于ID阶段的暂停机制。

10. lo\_hi\_r [2-bit]：指示是否需要读取高低寄存器（HI/LO）。

lo\_hi\_w [2-bit]：指示是否需要写入高低寄存器。

w\_hi\_we [1-bit]：写HI寄存器的使能信号。

w\_lo\_we [1-bit]：写LO寄存器的使能信号。

hi\_i [32-bit]：传递给HI寄存器的输入数据。

lo\_i [32-bit]：传递给LO寄存器的输入数据。

hi\_o [32-bit]：HI寄存器的输出数据。

lo\_o [32-bit]：LO寄存器的输出数据。

11. mul\_result [64-bit]：乘法运算的结果。

12. div\_result [64-bit]：除法运算的结果。

13. div\_ready\_i [1-bit]：除法运算是否完成的信号。

### 2.3.4 修改内容

（1）在EX段，对ex\_to\_id 进行数据通路连线，使用ex\_to\_id\_bus解决了第一个报错。

（2）在EX段，添加stallreq\_for\_ex信号及其数据通路。

（3）在EX段，添加inst\_stall需要的ready\_ex\_to\_id 控制信号。

（4）在EX段，添加data\_ram\_read相关的控制信号，为lw，lb等访存指令（point43）的数据读取做数据通路。

（5）在EX段，添加与hi，lo寄存器相关定义和数据通路，接入乘法和除法运算的输入输出参数。

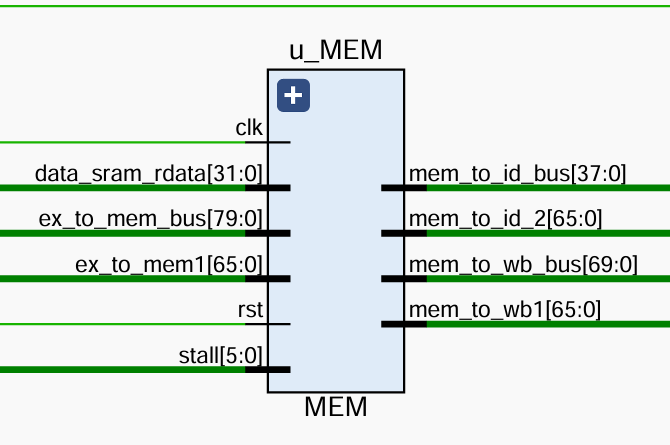
（6）修改乘法器符号判断bug，通过point 64。

## 2.4 MEM 段

### 2.4.1 整体功能介绍

该模块负责处理EX阶段与MEM阶段之间的信号传递，特别是内存读写操作以及寄存器堆的写回。该模块通过寄存器缓存（ex\_to\_mem\_bus\_r 和 ex\_to\_mem1\_r）接收来自EX阶段的数据，然后根据内存访问的相关信号，如内存使能（data\_ram\_en）、内存写使能（data\_ram\_wen）和选择寄存器堆数据的控制信号（sel\_rf\_res）来决定如何处理数据。通过适当的内存操作，最终将内存读取结果与EX阶段的计算结果进行合并，并准备好数据供下一阶段使用。该模块还具有调试信号和特殊寄存器（如HI、LO寄存器）的写回控制。

### 2.4.2 端口介绍



（1）输入端口：

data\_sram\_rdata[31:0],ex\_to\_mem\_bus[79:0],ex\_to\_mem1[65:0],

stall[5:0],rst, clk

（2）输出端口：

mem\_to\_id\_bus[37:0],mem\_to\_id\_2[65:0],mem\_to\_wb\_bus[69:0],

mem\_to\_wb1[65:0]

### 2.4.3 信号介绍

（1）clk：时钟信号，提供同步时序。

（2）rst：复位信号，用于重置内部状态。

（3）stall[5:0]：用于流水线暂停控制，stall[3]表示是否暂停当前指令的MEM阶段，stall[4]用于判断是否暂停MEM模块的操作。

（4）ex\_to\_mem\_bus[EX\_TO\_MEM\_WD-1:0]：来自EX阶段的数据总线，传递MEM阶段所需的控制信号和数据（如PC值、内存使能信号、内存写使能信号、计算结果等）。

（5）data\_sram\_rdata[31:0]：来自数据SRAM的读取数据，表示内存中的内容，供MEM阶段使用。

（6）mem\_to\_wb\_bus[MEM\_TO\_WB\_WD-1:0]：传递给WB阶段的数据总线，包括当前的PC值、寄存器写使能信号、寄存器地址和数据。

（7）mem\_to\_id\_bus[37:0]：用于向ID阶段传递寄存器写回的控制信息（如写使能信号、写寄存器地址、数据等）。

（8）ex\_to\_mem1[65:0]：来自EX阶段的额外数据总线，传递HI、LO寄存器的控制信息（如写使能信号及数据）。

（9）mem\_to\_wb1[65:0]：传递给WB阶段关于HI、LO寄存器的数据总线。

（10）mem\_to\_id\_2[65:0]：与mem\_to\_wb1相同的数据总线，传递HI、LO寄存器的数据和控制信号。

逻辑实现：

1. ex\_to\_mem\_bus\_r和ex\_to\_mem1\_r是寄存器，用于缓存从EX阶段传递过来的信号。当rst信号为高时，这些寄存器会被重置。stall[3]为Stop且stall[4]为NoStop时，数据会被清空，防止在流水线暂停时继续传递数据。只有当stall[3]为NoStop时，数据才会被传递到寄存器中，进行下一周期的处理。
2. mem\_pc、data\_ram\_en、data\_ram\_wen、sel\_rf\_res、rf\_we、rf\_waddr、ex\_result、data\_ram\_read从ex\_to\_mem\_bus\_r中解包，提供MEM阶段所需的控制信号和数据。mem\_pc表示当前指令的PC值，data\_ram\_en表示是否启用数据RAM，data\_ram\_wen表示内存写使能信号，sel\_rf\_res选择是否从内存读取数据写回寄存器堆，rf\_we和rf\_waddr分别表示寄存器写使能和目标寄存器地址，ex\_result表示EX阶段计算出的结果，data\_ram\_read表示内存读取时需要的字节选择信号。
3. w\_hi\_we、w\_lo\_we、hi\_i、lo\_i从ex\_to\_mem1\_r中解包，传递给WB阶段和ID阶段，指示是否写回HI、LO寄存器及其数据。
4. mem\_result表示从数据SRAM读取的结果，接收来自data\_sram\_rdata的内存数据。
5. rf\_wdata的计算取决于data\_ram\_read和data\_ram\_en的值，它决定了是将内存读取的结果（mem\_result）直接写回寄存器，还是根据内存访问的字节顺序进行截取和扩展，保证读取到正确的数据。
6. mem\_to\_wb\_bus由mem\_pc、rf\_we、rf\_waddr和rf\_wdata组成，最终传递给WB阶段。

7. mem\_to\_id\_bus用于向ID阶段传递寄存器堆写回的数据，包括寄存器写使能信号、寄存器地址和数据。

### 2.4.4 修改内容

（1）在MEM段，对mem\_to\_id 进行数据通路连线。

（2）在MEM段，添加data\_ram\_read相关的控制信号，为lw，lb等访存指令（point43）的数据读取做数据通路。

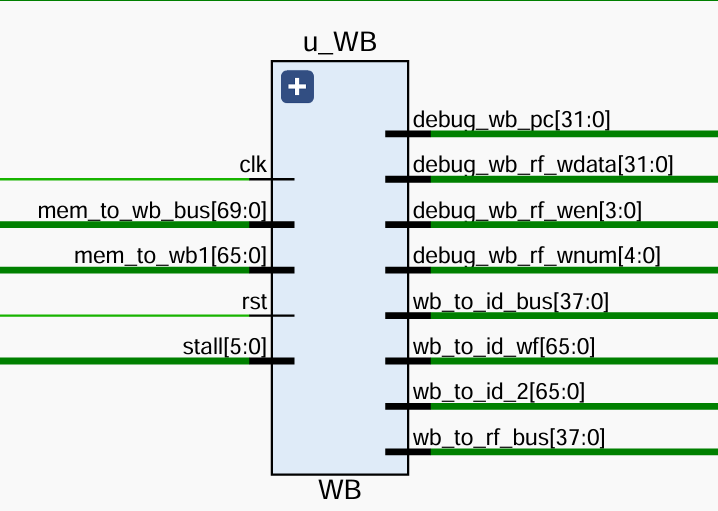
（3）在MEM段，添加与hi，lo寄存器相关定义和数据通路（mem\_to\_wb1和mem\_to\_id2）。

## 2.5 WB 段

### 2.5.1 整体功能介绍

该模块在处理流水线数据时，负责将来自MEM阶段的数据通过总线传递给WB阶段。通过适当的时序控制（如stall信号和rst信号），它确保数据在合适的时机被传递，避免了不必要的覆盖或冲突。同时，它还向其他模块（如ID阶段和RF模块）传递一些控制信号。模块中实现了数据寄存的功能，确保流水线在不同阶段的稳定性。此外，还提供了调试接口，便于在开发过程中查看WB阶段的状态。

### 2.5.2 端口介绍



（1）输入端口：

clk,rst,mem\_to\_wb\_bus[69:0],mem\_to\_wb1[65:0],stall[5:0]

（2）输出端口：

wb\_to\_id\_bus[37:0],wb\_to\_id\_wf[65:0],wb\_to\_id\_2[65:0],

wb\_to\_rf\_bus[37:0]

### 2.5.3 信号介绍

（1）clk：时钟信号，提供同步时序。

（2）rst：复位信号，用于重置内部状态。

（3）stall：[5:0]，用于流水线暂停控制，stall[4]表示是否暂停当前指令的WB阶段，stall[5]则用于判断是否暂停WB模块的操作。

（4）mem\_to\_wb\_bus[MEM\_TO\_WB\_WD-1:0]：来自MEM阶段的数据总线，传递WB所需的数据（如PC、写回寄存器地址、数据等）。

（5）mem\_to\_wb1[65:0]：额外的数据总线，传递一些特定控制信号（如HI、LO寄存器的写使能信号及其数据）。

（6）wb\_to\_rf\_bus[WB\_TO\_RF\_WD-1:0]：传递给寄存器堆的数据总线，包括写使能信号、写寄存器地址和写回数据。

（7）wb\_to\_id\_bus[37:0]：用于向ID阶段传递写回的控制信息，包括写使能信号、写寄存器地址和数据。

（8）debug\_wb\_pc[31:0]：调试接口，显示当前WB阶段的PC值。

（9）debug\_wb\_rf\_wen[3:0]：调试接口，显示当前WB阶段的寄存器写使能信号。

（10）debug\_wb\_rf\_wnum[4:0]：调试接口，显示当前WB阶段写回的寄存器地址。

（11）debug\_wb\_rf\_wdata[31:0]：调试接口，显示当前WB阶段的写回数据。

（12）wb\_to\_id\_wf[65:0]：额外的控制信号总线，传递给ID阶段关于HI、LO寄存器的信息（如写使能信号、数据）。

（13）wb\_to\_id\_2[65:0]：与wb\_to\_id\_wf相同的数据总线，传递HI、LO寄存器的数据和控制信号。

逻辑实现：

1. mem\_to\_wb\_bus\_r和mem\_to\_wb1\_r是寄存器，用于保存从MEM阶段到WB阶段传递的数据和控制信号。当rst为高时，这些寄存器会被重置。当stall[4]为Stop且stall[5]为NoStop时，数据会被清空，防止数据在暂停期间不被处理。当stall[4]为NoStop时，数据被传递到寄存器中，等待下一时钟周期的处理。
2. wb\_pc、rf\_we、rf\_waddr、rf\_wdata从mem\_to\_wb\_bus\_r中解包，提供给后续的WB阶段相关功能，主要用于寄存器堆的写回操作。
3. w\_hi\_we、w\_lo\_we、hi\_i、lo\_i从mem\_to\_wb1\_r中解包，传递给ID阶段，指示是否写回HI、LO寄存器及其数据。

4. debug信号提供了调试接口，用于在开发过程中查看WB阶段的状态。

### 2.5.4 修改内容

（1）在WB段，对wb\_to\_id 进行数据通路连线。

（2）在WB段，添加与hi，lo寄存器相关定义和数据通路（对mem\_to\_wb1进行解包并将数据打包到id）。

# 3. 实验感受及改进意见

## 3.1 孙毅

在完成MIPS指令CPU五段流水实验的过程中，我深刻体会到了计算机体系结构的精妙之处，同时也感受到了硬件设计与调试的难度。通过实现IF、ID、EX、MEM和WB五个流水段，我对CPU的工作原理有了更加深入的理解。

接下来总结一下为了完善这个流水线我们所做的工作：

1. 添加三个回传数据通路
2. 添加指令识别、操作数选择、pc选择的逻辑
3. 为了处理测试指令中存在的数据相关冲突，增添了暂停信号判断逻辑（新增stallreq\_for\_ex和stallreq\_for\_id）
4. 为了处理乘除法和四种数据迁移指令，添加hi、lo寄存器机器数据通路（包括正向传递和其他段对id段的回传两种），及其在三类指令中的应用逻辑（例如将乘法64位结果的高32位存入hi寄存器，低32位存入lo寄存器）

最后，调试过程让我认识到硬件设计的复杂性。由于流水线的各个阶段相互依赖，任何一个小的错误都可能导致整个系统的崩溃。通过逐步排查信号传递和时序问题，我逐渐掌握了硬件调试的技巧，也增强了对硬件设计的信心。

总的来说，这次实验让我从理论到实践全面理解了CPU流水线的工作原理，也让我认识到硬件设计的挑战与乐趣。未来，我希望能够进一步优化流水线的性能，探索更复杂的分支预测和数据前递机制，以实现更高效的CPU设计。

## 3.2 张硕昌

在本次设计中，我深入学习了CPU五级流水线的各阶段以及对应工作原理，对于五级流水线的工作流程有了更深入的了解,经过本次实验，我对课本上的知识更加的了解，同时更好的理解的流水线的具体功能。

在发布任务时，我们感到无从下手，因为没有系统学习过对应的编程语言，对于CPU五级流水线还停留在课本知识，所以在实际执行过程中我们遇到了很多困难，但是经过一段恶补学习以及不断地调试，我们也逐渐轻车熟路，逐步完善我们的代码。

在这个过程我很感谢我的同学们，没有他们的帮助我不可能这么快理解并完成对应的任务。这次实验对我帮助很大，因为是从零开始的代码语言学习，给了我很多未来学习的思路。

同时，在这里我想提一个建议，希望未来进行这门课程时能够提前讲解一些对应的代码知识，方便后来的同学尽可能轻松的完成实验。

# 4. 参考资料

1. 雷思磊.《自己动手做cpu\_雷思磊》[M/CD].
2. 《“系统能力培养大赛”MIPS指令系统规范\_v1.01》[M/CD].