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班

组长：詹昌明

组员：方嘉璐

组员：谢北

报告日期：2025.1.4

**目录**

**[一、 实验概要 3](#_Toc2312)**

[1. 个人工作量 3](#_Toc3334)

[2. 总体设计 3](#_Toc22479)

[2.1 取指令周期（IF段） 4](#_Toc15513)

[2.2 指令译码周期（ID段） 4](#_Toc8826)

[2.3 执行/有效地址计算周期（EX段） 4](#_Toc18696)

[2.4 访存/分支完成周期（MEM段） 4](#_Toc19971)

[2.5 写回周期（WB段） 4](#_Toc4500)

[3. 指令集 4](#_Toc23046)

[4. 程序运行环境及使用工具 4](#_Toc26411)

**[二、 详细说明 4](#_Toc27774)**

[1. IF段 4](#_Toc30129)

[1.1 功能说明 4](#_Toc28738)

[1.2 端口介绍 4](#_Toc22786)

[1.3 信号介绍 4](#_Toc1178)

[2. ID段 4](#_Toc17076)

[2.1 功能说明 4](#_Toc11887)

[2.2 端口介绍 4](#_Toc32533)

[2.3 信号介绍 4](#_Toc32721)

[2.4 Regfile 寄存器堆说明 4](#_Toc23298)

[3. EX段 4](#_Toc20723)

[3.1 功能说明 4](#_Toc14725)

[3.2 端口介绍 4](#_Toc15824)

[3.3 信号介绍 4](#_Toc24416)

[3.4 mul\_plus功能模块说明 4](#_Toc30218)

[4. MEM段 4](#_Toc27630)

[4.1 功能说明 5](#_Toc6348)

[4.2 端口介绍 5](#_Toc29181)

[4.3 信号介绍 5](#_Toc25784)

[5. WB段 5](#_Toc30865)

[5.1 功能说明 5](#_Toc1049)

[5.2 端口介绍 5](#_Toc10931)

[5.3 信号介绍 5](#_Toc16386)

[6. CTRL模块 5](#_Toc30387)

[6.1 功能说明 5](#_Toc30969)

[6.2 端口介绍 5](#_Toc8330)

[6.3 信号介绍 5](#_Toc31325)

**[三、 实验感受及改进建议 5](#_Toc10304)**

[1. 詹昌明 5](#_Toc9650)

[2. 方嘉璐 5](#_Toc11390)

[3. 谢北 5](#_Toc7527)

**[四、 参考资料 5](#_Toc29364)**

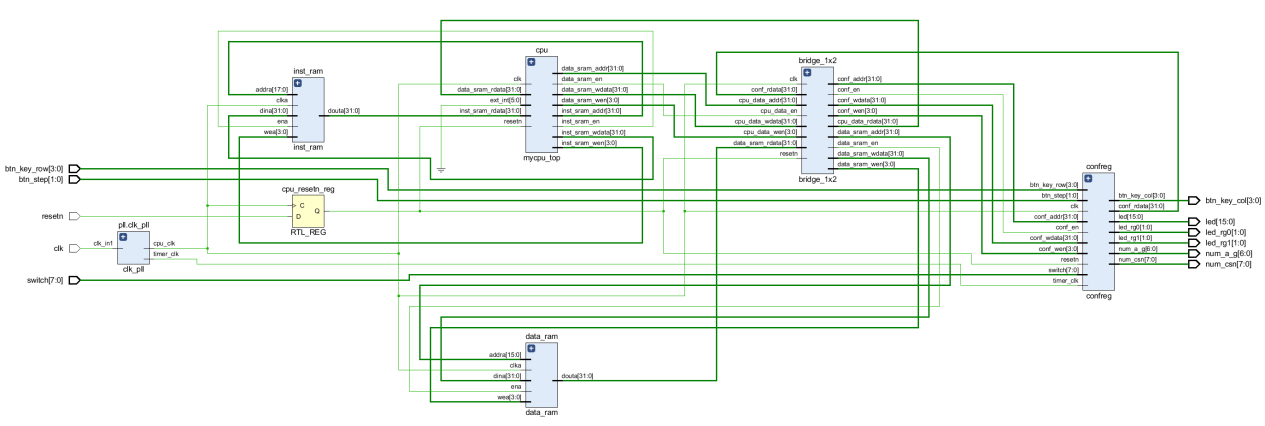
# 实验概要

## 个人工作量

詹昌明 70% 方嘉璐15% 谢北15%

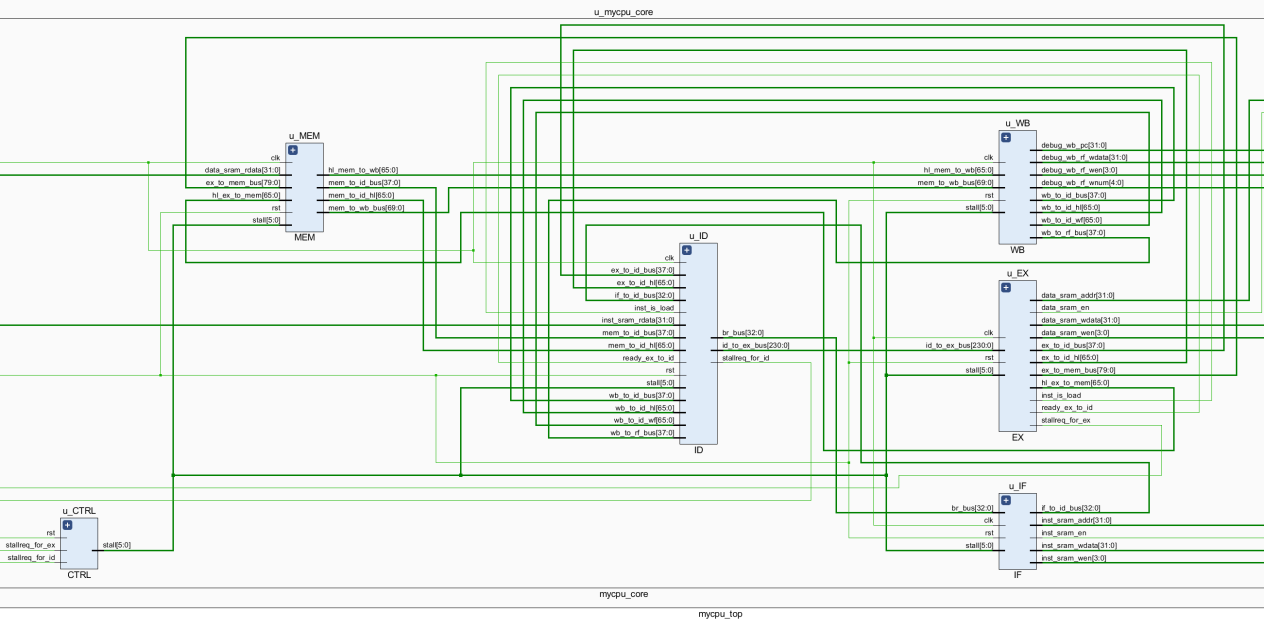
## 总体设计

总体电路的顶层设计如下：

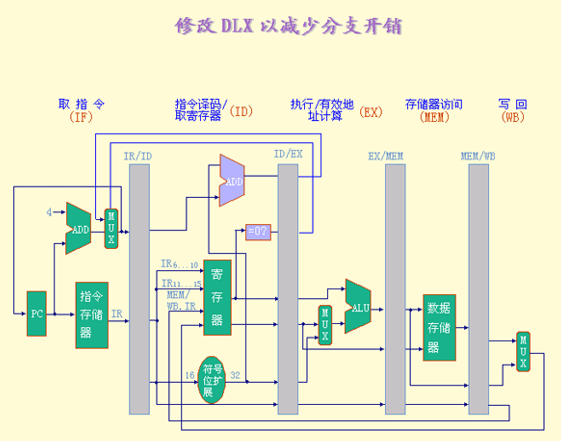


该 CPU 的顶层结构包括输出时钟周期信号的 pll.clk\_pll 部分、指令存储器 inst\_ram 部分、数据存储器 data\_ram 部分、输出复位信号的 cpu\_resetn\_reg 部分、cpu 部分、内存虚地址映射 bridge\_1x2 部分、外部接口confreg 部分，通过各部分的输入输出端口、使能信号实现交互。整个系统的功能清晰且便于扩展。

不同流水段之间的连接图如下：



我们将整体的处理过程分为取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器回写（WB）五级，对应多周期的五个处理阶段。其中，一个指令的执行需要 5 个时钟周期，每个时钟周期的上升沿来临时，此指令所代表的一系列数据和控制信息将转移到下一级处理。DLX的基本流水线示意图如下所示：



### 取指令周期（IF段）

IR ← Mem[PC]

NPC ← PC＋4

该周期的功能是从指令存储器中取出指令。

本段有一个 PC 寄存器用于存放指令地址，PC输入是一个选择器选择的结果。从 PC 寄存器中取出指令地址，根据PC从指令存储器中取出指令，送入IF/ID流水段寄存器以便后面流水段使用，并把PC+4的值一并存入流水段寄存器。这是用于分支指令的控制冒险用于恢复PC所用。

### 指令译码周期（ID段）

A ← Regs[IR6 ..10]

B ← Regs[IR11 ..15]

Imm ← (IR16)16 ## IR16 ..31

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，译码完成后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种，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。

### 执行/有效地址计算周期（EX段）

不同指令在该周期完成的工作也不一样，主要可以分为以下五种：

1. 寄存器―寄存器

ALUOutput ← A op B

1. 寄存器―立即值

ALUOutput ← A op Imm

1. 存储器访问（load 和 store）

ALUOutput ← A＋Imm

1. 计算偏移地址

Cond ← (A op 0) 判断分支是否成功，失败则结束

1. 分支操作

ALUOutput ← NPC＋Imm

在EX段，流水线根据指令的编码进行算术运算、逻辑运算或计算条件分支指令的跳转目标地址。具体来说，对于LW（Load Word）和SW（Store Word）指令，EX段会计算RAM的访问地址，该地址由寄存器中的值与指令中的偏移量相加得到。数据运算是从上一流水段（ID段）的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相应的运算，并将运算结果存入下一个流水段（MEM段）的寄存器中。

此外，EX段还负责处理跳转和分支指令的地址计算。如果指令是跳转或分支指令，EX段会计算跳转目标地址，并将该地址送回IF段（取指阶段）的PC（程序计数器）选择器，作为下一个指令的地址来源。跳转指令的结果会在下一个流水段（MEM段）进行判断，以确定是否需要真正执行跳转。这是因为在EX段无法立即确定跳转条件是否满足，只有在MEM段才能最终判断跳转是否有效。

### 访存/分支完成周期（MEM段）

1. 存储器访问（load 和 store）

LMD ← Mem[ALUOutput] 或 Mem [ALUOutput] ← B

1. 分支操作

if（cond）PC ← ALUOutput 成功则把计算好的地址放入PC

else PC ← NPC 否则跳过

其他类型指令均跳过该阶段

MEM段主要负责内存相关的操作。只有在执行LW和SW指令时，才会对存储器进行实际的读写操作。对于其他指令，MEM段仅起到一个周期的传递作用，即将EX段的结果直接传递给WB段，而不进行内存访问。

在内存的读写操作之前，设计了一个移位器，专门为LW和SW等指令服务。移位器的作用是根据指令的要求，对内存地址进行适当的调整，以确保数据的正确读取或写入。此外，MEM段还包含一个条件检查单元，用于判断跳转指令是否有效。该单元会根据EX段计算的条件结果，决定是否需要执行跳转操作，并判断是否真的需要将结果写回寄存器。这样能确保跳转指令的正确执行，并避免了不必要的寄存器写回操作。

### 写回周期（WB段）

1. 寄存器―寄存器型 ALU 指令

Regs[IR16 ..20] (rd)← ALUOutput

1. 寄存器―立即值型 ALU 指令

Regs[IR11 ..15] (rt)← ALUOutput

1. Load 指令

Regs[IR11 ..15] (rt)← LMD

在WB段（写回阶段），指令执行的结果会被写回到寄存器文件中。具体来说，ALU运算指令的结果数据直接来自ALU的计算结果，而load指令的结果数据则来自存储器。这两种指令的结果都会在这个阶段被写入通用寄存器组。

整个流水线CPU的时钟设计非常关键。每个流水段寄存器的写入操作发生在时钟的下降沿，而寄存器的读取操作则发生在时钟的上升沿。这种设计是为了解决潜在的结构冲突问题，即当某个寄存器在同一时钟周期内既被写入又被读取时，可能会导致数据不一致。通过将读写操作分配到不同的时钟边沿，可以有效避免这种冲突。

然而，这种设计也带来了一个新的问题：由于写入操作发生在下降沿，而读取操作发生在上升沿，可能会导致数据延迟。也就是说，当前指令需要读取的数据可能还没有被前一条指令写回寄存器。为了解决这个问题，CPU设计中引入了课上所讲的数据定向技术。数据定向允许将尚未写回寄存器的结果直接传递给后续指令，从而避免因数据延迟而导致的错误。这种机制确保了流水线的高效运行，同时避免了结构冲突和数据竞争问题。

## 指令集

我们添加完成了如下指令：

ori, lui, addiu, beq, subu, jr, jal, addu, bne, sll, or, lw, sw, xor, sltu, slt, slti, sltiu, j, add, addi, sub, and, andi, nor, xori, sllv, sra, bgez, bltz, bgtz, blez, bgezal, bltzal, jalr, mflo, mfhi, mthi, mtlo, div, divi, mult, multu, lb, lbu, lh, lhu, sb, sh

包括算术运算指令、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。成功通过第 64 个点，并制作了一个 32 时钟周期的乘法器。

## 程序运行环境及使用工具

操作系统：Windows 11/10。

开发平台：Vivado 2019.2。

编程语言：Verilog HDL 硬件描述语言。

# 详细说明

## IF段

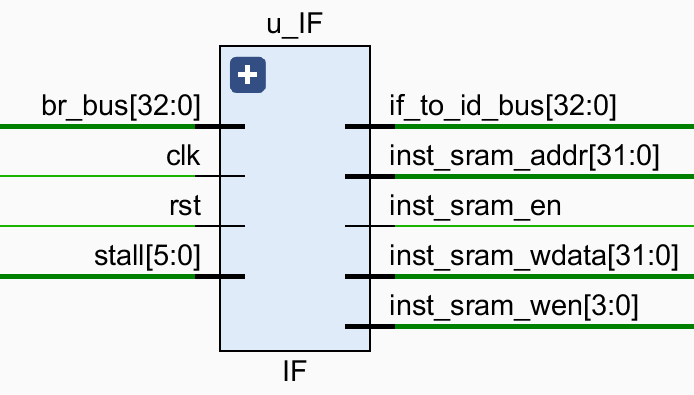
### 功能说明

在IF段，程序计数器（PC）中的值被用作地址，从指令存储器中取出对应的指令，并将其放入指令寄存器（IR）中。与此同时，PC的值会自动加4（每条指令占4个字节），指向下一条顺序指令的地址。

IF段的主要任务是确定下一个PC的值。IF段将当前PC值传递给指令存储器。 在ID段，指令存储器根据PC值返回对应的32位指令（inst），供ID段译码。同时，IF段将PC值传递给ID段，便于后续指令的执行。这种设计确保了指令的连续获取和正确译码，为流水线的顺利运行奠定了基础。

若当前指令是跳转指令，IF段还会根据跳转目标地址更新PC；否则，PC会继续顺序递增。此外，IF段还会将取指令信号置为真，并将相关的总线信号传递到WB段。

### 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\_di\_bus[32:0], inst\_sram\_addr[31:0], inst\_sram\_en, inst\_sram\_wdata[31:0],inst\_sram\_wen[3:0]

### 信号介绍

rst 是接收到的复位信号，用于对系统进行初始化，clk 是接收到的时钟信号，用于同步各模块的操作。

br\_bus[32:0] 是从 ID 阶段传递过来的跳转控制信号，用于改变下一条指令的地址。其中包含了 br\_e（跳转使能信号）和 br\_addr[31:0]（跳转目标地址）。当 br\_e 信号为 1，且 br\_addr[31:0] 有效时，将 br\_addr[31:0] 的值赋给当前指令的 PC（程序计数器），并将这个 PC 值发送到指令寄存器，从而在 ID 阶段确定跳转后的指令。

stall[5:0] 是从 CTRL.v 模块接收到的暂停控制信号。如果 stall[0] 为 1（即 stall[0] == 1'b1），则保持当前 PC 的值不变，实现暂停操作，使流水线停顿以等待数据准备完成。如果没有暂停信号，PC 将正常更新，即执行加 4 操作（顺序执行下一条指令）或者根据跳转信号更新为跳转地址。

if\_to\_id\_bus[32:0] 用于将当前阶段计算得到的 PC 值传递到 ID 阶段，供后续指令译码使用。

inst\_sram\_en 是指令存储器的使能信号，控制指令存储器的读取操作。

inst\_sram\_addr[31:0] 保存了当前 PC 的值，并将其发送到指令存储器。指令存储器在接收到 PC 地址后，根据该地址读取对应的指令，并将指令值（inst）传递到 ID 阶段进行指令译码和执行。

inst\_sram\_wdata[31:0] 和 inst\_sram\_wen[3:0] 是用于向指令存储器写入数据的信号。然而，在 IF 阶段（取指阶段），不涉及对指令存储器的写操作，因此这两个信号在该阶段被固定为 0。这也是为了保持流水线的统一性、灵活性和可扩展性，并为后续的设计扩展提供便利。

## ID段

### 功能说明

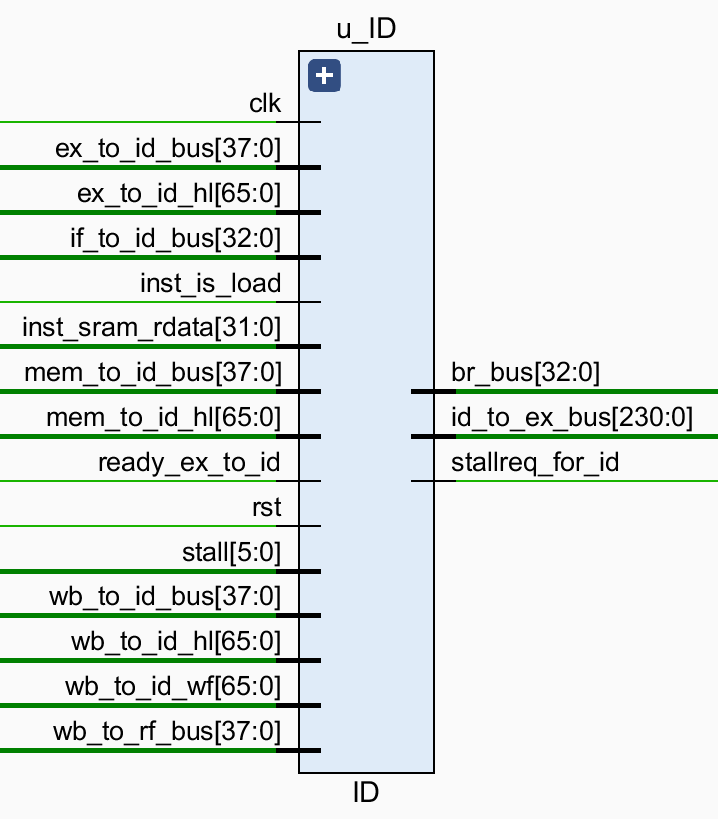
对从指令存储器中得到的指令 inst 值进行译码，并用 IR 中的寄存器地址去访问通用寄存器组，从 32 个通用寄存器 reg\_array[31:0]中读出要使用的寄存器的值。

如果指令中含有立即数，那么还要将立即数进行符号扩展或无符号扩展。如果是转移指令满足转移条件，那么要给出转移目标，作为新的指令地址。若解析到指令为跳转指令，则计算跳转地址，并将结果通过 br.bus，总线传回 IF 段，以便 IF 段在最短时间拿到下一条指令的地址。另外将各种使能信号以及操作数 1 和操作数 2 通过 id\_to\_ex 总线传递给下一阶段的 EX。

就算将简单的分支跳转判断前推到ID段完成，流水线还是会产生一个时钟周期的延迟，这时就需要用到我们学过的延迟槽技术了。在分支指令之后紧跟的一条或几条指令称为延迟槽，无论分支指令最终是否跳转，延迟槽中的指令都会被执行。这是利用分支指令执行的延迟时间，填充一些有用的指令，避免这段时间处理器空闲。

### 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\_hl[65:0],mem\_to\_id\_hl[65:0],wb\_to\_id\_hl[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



### 信号介绍

rst 是接收到的复位信号，clk 是接收到的时钟信号。

if\_to\_id\_bus[32:0]是从 IF 传过来的当前流水线所处理的 pc 值，并将其[31:0]的值附给 inst。inst\_sram\_rdata[31:0]是从指令存储器接收到的当前pc值对应指令的inst值， 当 ID 段没有被暂停时，则将此值赋给 inst 变量并进行指令判断，找出 inst 所对应的指令，以备进行后续的指令操作。

ex\_to\_id\_bus[37:0],mem\_to\_id\_bus[37:0],wb\_to\_id\_bus[37:0],ex\_to\_id\_hl[65:0],mem\_to\_id\_hl[65:0],wb\_to\_id\_hl[65:0]这些指令都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 ex 段、mem 段、wb 段提前发给ID 段，再由 ID 段发送给 regfile.v 文件中，进行赋值给 rs 和 rt 所需要的寄存器的值。

stall[5:0]是从 CTRL.v 文件中接收到的暂停信号，如果 stall[2]==1’b1，则将寄存器 if\_to\_id\_bus\_r 的值赋值为 0，以来保证从 if 段传过来的值不能在当前时钟周期赋值，则保证了下一个时钟周期不会得到从当前时钟周期传入到下一个阶段的值，从而将当前时钟周期的操作暂停。



同时将当前时钟周期的 inst 保存在临时寄存器 inst\_stall[31:0]中，同时将该寄存器的使能信号 inst\_stall\_en 赋值为 1’b1。并在下一个时钟周期将该寄存器的值赋值给下一个时钟周期的 inst 值，在 stall[2]==1’b0 后则将该寄存器的值变为32’b0，其该寄存器的使能信号 inst\_stall\_en 也变为 1’b0。以此来达到暂存当前时钟周期的 inst 的目的。

ready\_ex\_to\_id 是从 EX 段传过来的的一个信号，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。

wb\_to\_id\_wf[65:0]wb\_to\_rf\_bus[37:0]是从 WB 进行写回寄存器的值，在 ID 段中进行接收，并将其直接传入到regfile.v 文件中去，从而进行写寄存器的操作。

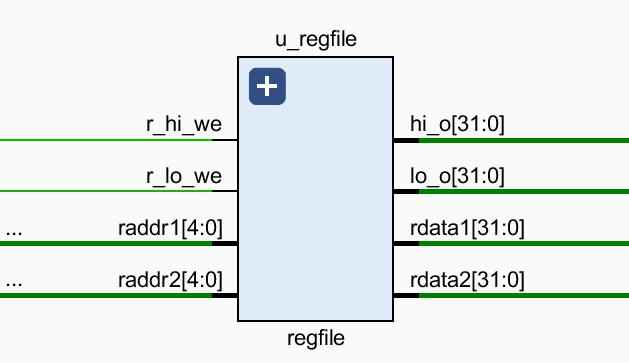
inst\_is\_load 是从 EX 段接收到的信号，用来判断 EX 的当前指令是否是 LW 指令，如果是，则该值为 1，如果不是，则该值为 0。并且与 rs 寄存器和 rt 寄存器中的地址进行判断，如果 EX 段的 lw 要写入的寄存器的地址与当前 ID 段的指令要读取的寄存器的地址有相同的，则 stallreg\_for\_id 为 1’b1，并将此值赋值给 CTRL.v 中， 在此文件中发出暂停 ID 段和 IF 的暂停信号。

br\_bus[32:0]是跳转指令所要跳转到的 pc 值，将跳转后的 pc 值和跳转使能信号发送给 ID 段，从而让下一个流水线进行跳转到目标指令。

id\_to\_ex\_bus[230:0]是 ID 段要发送给 EX 的值，其中包括了：当前指令的 pc 值和 inst 值，进行 alu\_op 操作的信号，进行 alu操作数 1 的目标值选择 sel\_alu\_src1 和进行 alu 操作数 2 的目标值选择 sel\_slu\_src2，对存储器进行访存操作的使能信号 data\_ram\_en，对存储器进行操作的 data\_sram\_wen[3:0],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，寄存器 rs和寄存器 rt 所存的数值 rdata1 和 rdata2，对 hilo 寄存器进行读写操作的 lo\_hi\_r[1:0]和 lo\_hi\_w[1:0]，寄存器 hilo 所对应的数值 lo\_o 和 hi\_o，对寄存器进行操作的 data\_ram\_read[3:0]。

### Regfile 寄存器堆说明

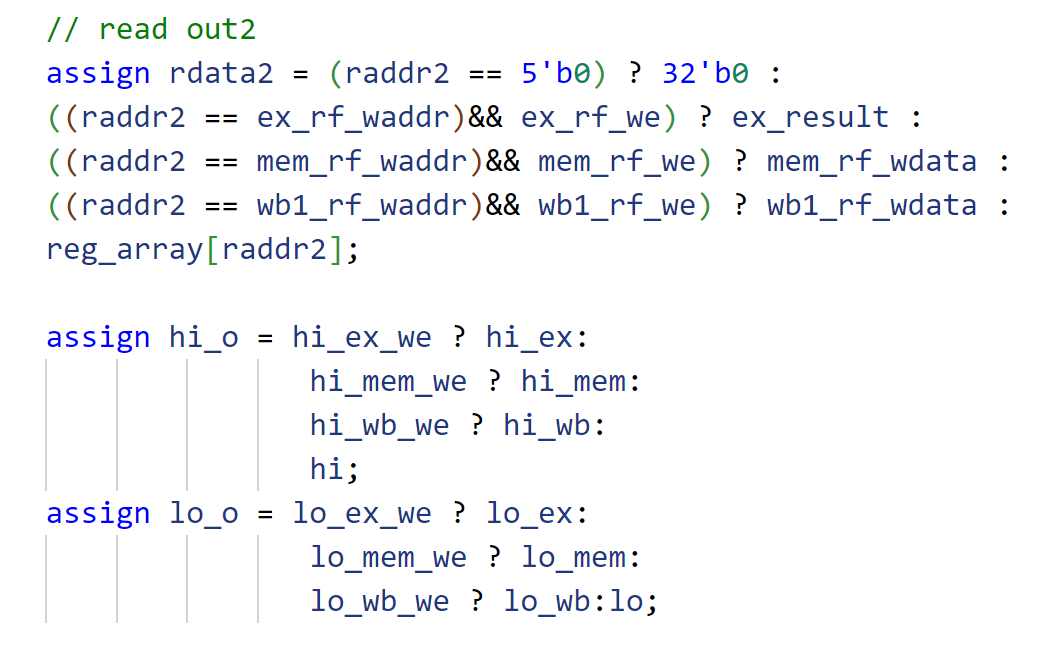
文件中包括了寄存器reg\_array[31:0]和存乘除法结果的 hilo 寄存器。



1. 输入信号：r\_hi\_we,r\_lo\_we,raddr1[4:0],raddr2[4:0]
2. 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，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。在 regfile.v 中，在这里需要解决数据相关的问题，将从 EX、MEM、WB 段需要写到寄存器的值提前传入到 regfile.v 中，如果当前指令中 rs 和 rt 的值与即将要写入寄存器的地址相同，则将传回要写入寄存器的值提前赋给 rdata1[31:0]或者rdata2[31:0],从而避免了读取到的寄存器值不正确的情况，从而解决了数据相关的问题，代码如下：



## EX段

### 功能说明

按照译码阶段给出的操作数、运算类型，进行运算，给出运算结果。如果是load/store 指令，那么还会计算 load/store 的目标地址。在这个周期，不同的指令有不同的操作。

1. **load 指令**

访存有效地址：Regs[rs]＋immediate

ALU 把指令中所指定的寄存器的内容与偏移量相加，形成访存有效地址，将从存储器取来的数据放入寄存器 rt 中。

1. **store 指令**

访存有效地址：Regs[rs]＋immediate

ALU 把指令中所指定的寄存器的内容与偏移量相加，形成访存有效地址，将要存入存储器的数据放在寄存器 rt 中。

寄存器－寄存器 ALU 指令

ALU 按照操作码指定的操作对从通用寄存器组中读出的数据计算。

寄存器－立即数 ALU 指令

ALU 按照操作码指定的操作对从通用寄存器组中读出的操作数和指令中给出的立即数进行运算。

1. **立即数指令**

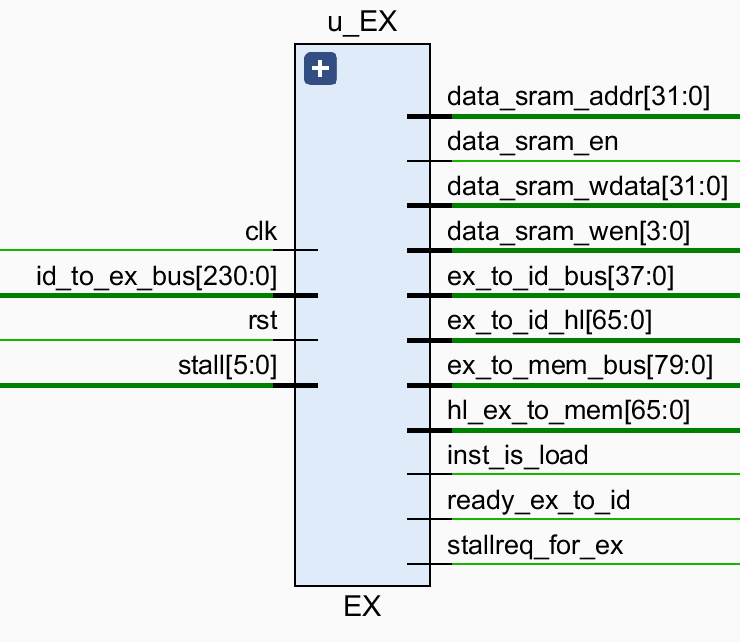
Regs[rt] ← Regs[rs] op immediate

1. **分支指令**

转移目标地址：Regs[rs]＋immediate，rt 无用

ALU 把指令中给出的偏移量与 PC 值相加，形成转移目标的地址。同时，对在前一个周期读出的操作数进行判断，确定分支是否成功。寄存器跳转、寄存器跳转并链接转移目标地址为 Regs[rs]。

### 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],hl\_ex\_to\_mem[65:0],inst\_is\_load,ready\_ex\_to\_id,stall\_for\_ex

### 信号介绍

rst 是接收到的复位信号，clk 是接收到的时钟信号。

id\_to\_ex\_bus[230:0]是 ID 段要发送给 EX 的值，其中包括了：当前指令的 pc 值和 inst 值，进行 alu\_op 操作的信号，进行 alu 操作数 1 的目标值选择 sel\_alu\_src1和进行 alu 操作数 2 的目标值选择 sel\_slu\_src2，对存储器进行访存操作的使能data\_ram\_en，对存储器进行操作的 data\_sram\_wen[3:0],对reg\_array[31:0]进行写操作的使能信号 rf\_we,对reg\_array[31:0]进行写操作的寄存器的地址，寄存器 rs 和寄存器 rt 所存的数值 rdata1 和 rdata2，对 hilo 寄存器进行读写操作的 lo\_hi\_r[1:0]和 lo\_hi\_w[1:0]，寄存器 hilo 所对应的数值 lo\_o 和 hi\_o，对寄存器进行操作的 data\_ram\_read[3:0]。

stall[5:0]是从 CTRL.v 文件中接收到的暂停信号，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，所以需要将 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。

data\_sram\_en 时对存储器的访存使能信号，当需要对存储器进行操作时，则令其值为 1’b1。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\_wen [3:0]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\_mem\_bus[79:0]是 EX 段要发送给 MEM 的值，其中包括了：当前指令的 pc 值，对存储器进行访存操作的使能信号 data\_ram\_en，对存储器进行操作的寄存器data\_sram\_wen[3:0],对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 段中算出的结果ex\_result[31:0],对寄存器进行操作data\_ram\_read[3:0]。

hl\_ex\_to\_mem[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。

ex\_to\_id\_hl[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。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。

ex\_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，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。

### mul\_plus功能模块说明

自制乘法器说明步骤：

1.将操作数 1 和操作数 2 进行取绝对值操作，如果其首位为 1 且当前乘法为有符号乘法，则将此操作数取反后加 1 作为原操作数的绝对值。

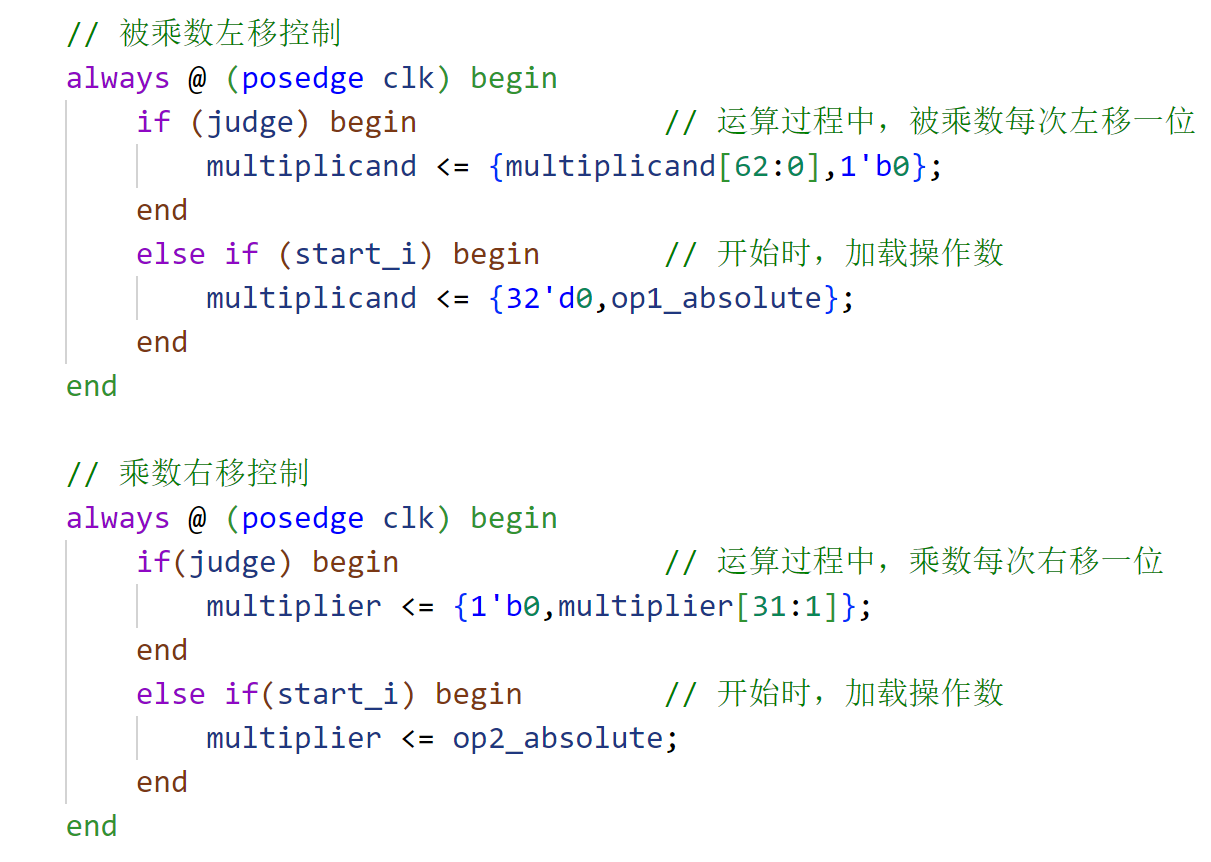
2.将操作数 1 的绝对值作为被乘数，其[63:32]位赋值为 0，[31:0]位赋值位操作数 1 的绝对值，再将操作数 2 作为乘数[31:0];

3.每经过一个时钟周期，被乘数左移一位，最低位补 0，最高位舍弃；4.每经过一个时钟周期，乘数右移一位，最高位补 0，最低为舍弃；

5.如果当前时钟周期的乘数最低位为1，则将当前时钟周期的被乘数与乘法的临时结果相加赋值给临时结果，如果为0，临时结果保持不变；

6.如果乘数不为 0，则跳回到 3，如果乘数为 0，则结束循环；

7.如果是有符号乘法，则将原操作码 1 和 2 的符号位进行取非或，得到最终结果的符号，如果是有符号的，则将得到的临时结果取反加 1，否者临时结果直接赋值给输出。



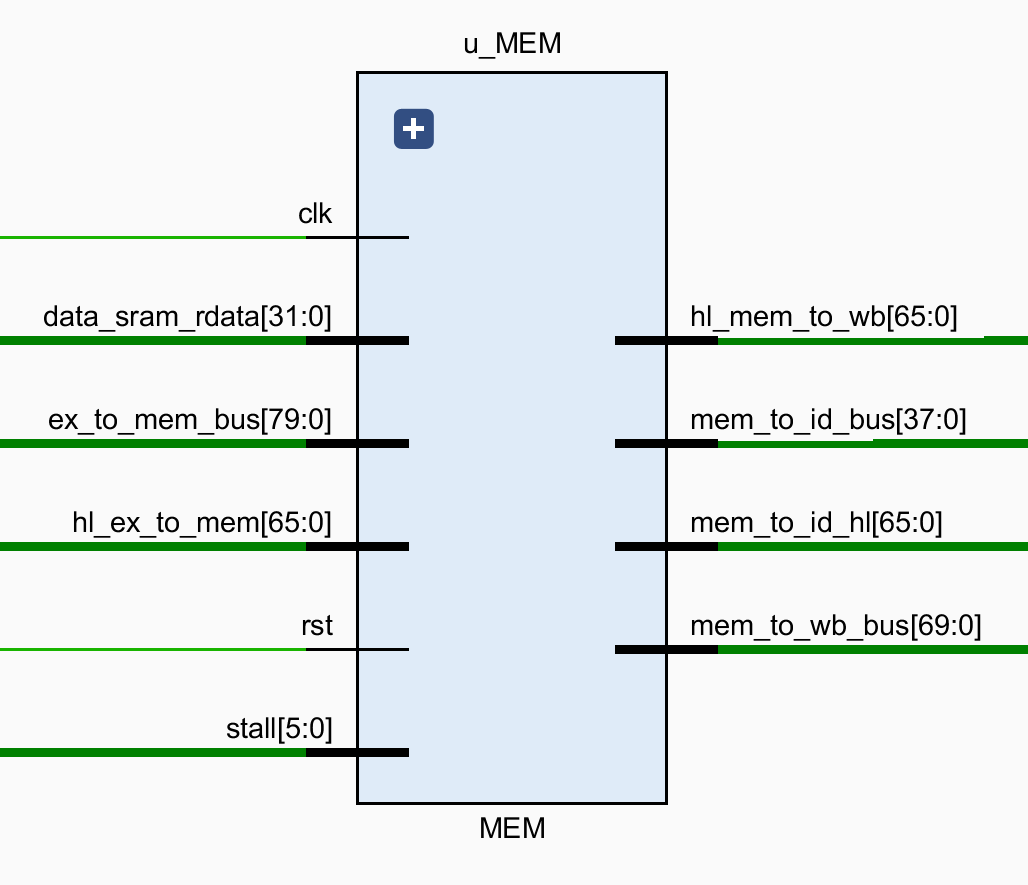


## MEM段

### 功能说明

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。

### 端口介绍



1. 输入端口：data\_sram\_rdata[31:0],ex\_to\_mem\_bus[79:0], hl\_ex\_to\_mem[65:0],stall[5:0],rst, clk
2. 输出端口： mem\_to\_id\_bus[37:0],mem\_to\_id\_hl[65:0], mem\_to\_wb\_bus[69:0],hl\_mem\_to\_wb[65:0]

### 信号介绍

rst 是接收到的复位信号，clk 是接收到的时钟信号。

data\_sram\_rdata[31:0]是从存储器中在 EX 读取到的数值，在 MEM 接收。

ex\_to\_mem\_bus[79:0]是 MEM 段接收到的从 EX 段发送过来的值，其中包括了：当前指令的 pc 值，对存储器进行访存操作的使能信号 data\_ram\_en，对存储器进行操作的 data\_sram\_wen[3:0],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 段中算出的结果 ex\_result[31:0],对寄存器操作的data\_ram\_read[3:0]。

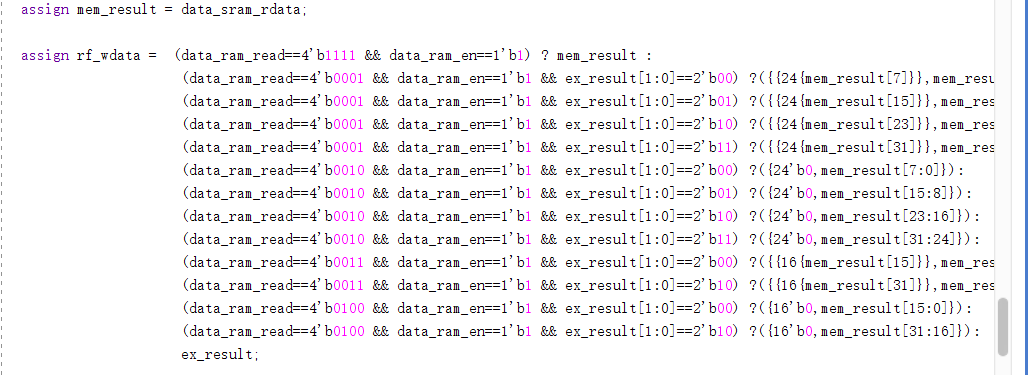
hl\_ex\_to\_mem[65:0]是 MEM 接收到从 EX 段发送过来的值，其包括了写 hi 和 lo 寄存器的使能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo 寄存器的值，如果不写，则此处为 0，且使能信号为 0。

mem\_to\_id\_hl[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。 hl\_mem\_to\_wb[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。

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\_wb\_bus[69:0]是 MEM 段要发送给 WB 段的值，其中包括了：当前指令的 pc 值，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 段中算出的结果 ex\_result[31:0]。

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。

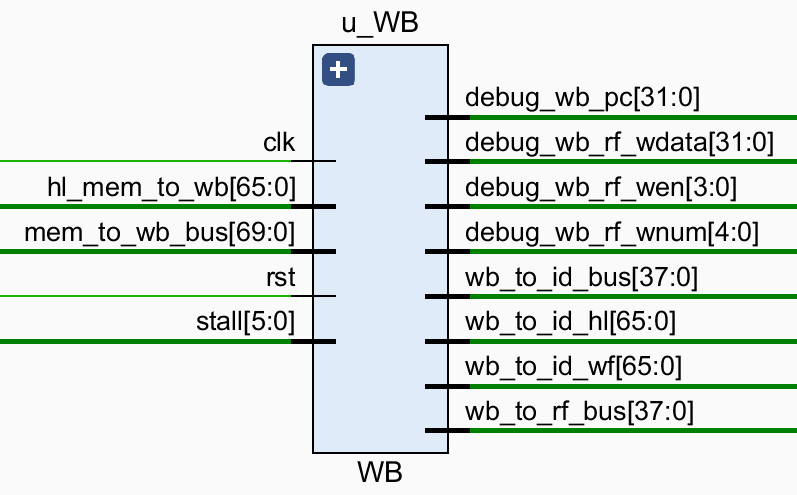


## WB段

### 功能说明

WB（写回）段是流水线的最后一个阶段，负责将来自EX段或MEM段的运算结果保存到目标寄存器中。通过将计算结果写回寄存器堆（reg\_array[31:0]），WB段完成了指令的执行。此外，如果涉及HI和LO寄存器的操作，WB段还负责将结果写入到相应的HI/LO寄存器中。确保计算结果准确写入寄存器堆是流水线执行的重要一步。

### 端口介绍



1. 输入端口：clk,rst,mem\_to\_wb\_bus[69:0],hl\_mem\_to\_wb[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\_hl[65:0], wb\_to\_rf\_bus[37:0]

### 信号介绍

rst 是接收到的复位信号，clk 是接收到的时钟信号。

mem\_to\_wb\_bus[69:0]是 WB 段接收到的从 MEM 段发过来的值，其中包括了：

当前指令的 pc 值，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 段中算出的结果ex\_result[31:0]。

hl\_mem\_to\_wb[65:0]是 WB 段接收到的从 MEM 段发过来的值，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。

wb\_to\_id\_hl[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。

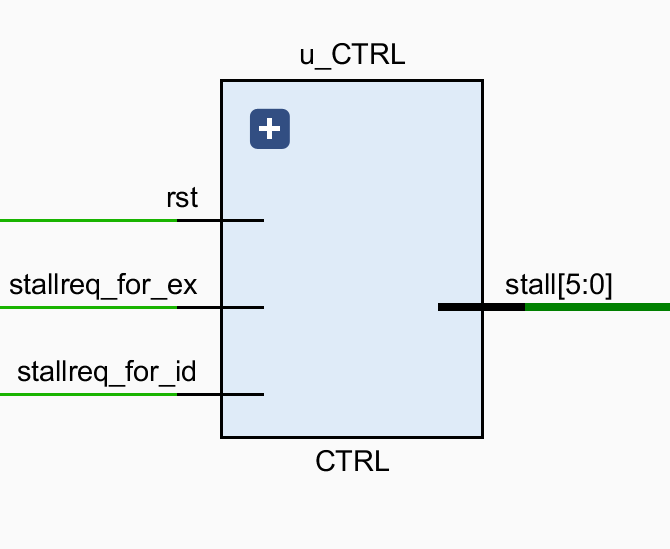
wb\_to\_id\_bus[37:0]是跟数据相关有关的指令，当当前指令需要取前面还未存入寄存器的值的时候，由 WB 段提前发给 ID 段，再由 ID 段发送给 regfile.v 文件中，进行赋给 rs 和 rt 所需要的寄存器的值。

wb\_to\_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 段中算出的结果 ex\_result [31:0]。

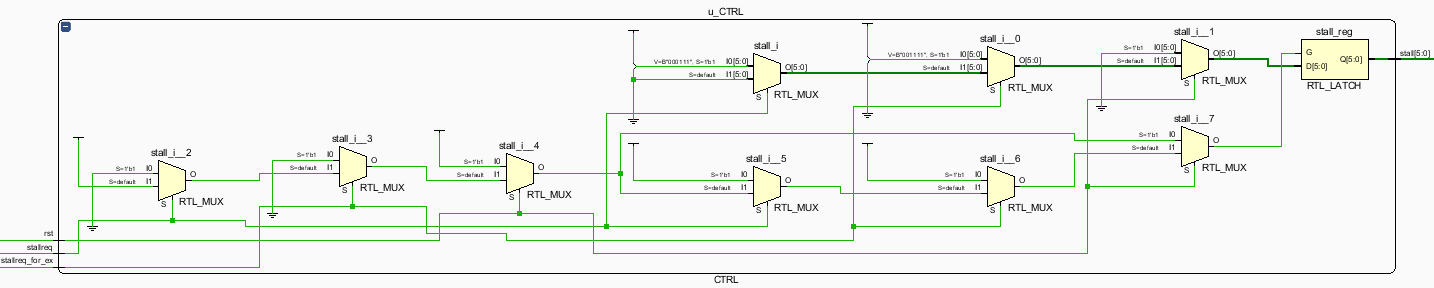
wb\_to\_id\_wf[65:0] 是 WB 段要发送给 regfile.v 的值，其包括了写 hi 和 lo 寄存器的使能信号，用于判断是否进行写寄存器的操作，还有包括了将要写入 hi 和 lo寄存器的值，如果不写，则此处为 0，且使能信号为 0。

## CTRL模块

### 端口介绍



1. 输入端口：rst，stallreq\_for\_ex,stallreq\_for\_id
2. 输出端口：[`StallBus-1:0] stall



### 设计原理

输出暂停信号：

stall[0]为 1 表示PC计数器暂停

stall[1]为 1 if 段暂停

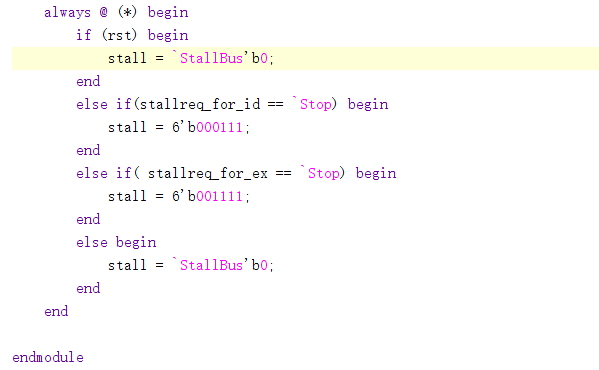
stall[2]为 1 id 段暂停

stall[3]为 1 ex 段暂停

stall[4]为 1 mem 段暂停

stall[5]为 1 wb 段暂停

如果rst为真（高电平），则将stall信号设置为全0。这表示在复位状态下不会产生任何阻断（stall）。如果stallreq\_for\_id等于`Stop，则将stall信号设置为6'b000111。这是一个特定的stall模式，对应于ID阶段的某些条件。如果stallreq\_for\_ex等于`Stop，则将stall信号设置为6'b001111。这是另一个特定的stall模式，对应于EX阶段的某些条件。以上两个条件均不满足，则stall信号被设置为全0，表示没有stall。



# 实验感受及改进建议

## 詹昌明

此次实验让我对 CPU 有了更深的理解，而不再只是局限于理论知识。

在编写流水线 CPU 的过程了，因为接触较少且很多地方需要自行设计，所以 许多看起来较为简单的问题，因为思维不够缜密，在执行的时候也出现了很多的困难。例如数据相关的解决，流水线暂停的实现，内存的读写的实现等问题，为此我通过查阅了资料和与队友交流，才解决了这个问题。在进行流水线暂停的设计时，一开始我并没有了解它真正的原理，于是设计的出来有 bug 无法正常运行，后期通过在网上查阅资料并咨询电子和计科专业的学霸，了解到了是指令值没有处理好，设计的程序会出现错误，我也逐一进行分析与修改，使我积累了很多的经验。

整个过程还是比较吃力的，但这次的实验也使我的对流水线的理解有了很大提高，而且是我更加深刻的理解了团队合作以及总体架构的重要性。在较短的时间内想要完成较多的任务，只有通过团队合作的方式。而要想使团队合作能够顺利进行，就需要使用模块化的的设计方法，合理安排软件的结构。总体上说，这次课程设计加深了我对理论的理解，增强了我的编码能力，让我积累了更多的有关于团队合作的经验，使我受益匪浅。最后得感谢助教的悉心指导和队友的支持。

## 方嘉璐

一开始我对vivado完全不了解，但通过在互联网和从书中慢慢摸索后，我逐渐从盲目困惑一步步走向正轨。例如，使用多路选择器实现对不同分支指令的不同跳转。通过实验的学习之后，我获得了深入的计算机体系结构知识，提高了我的团队协作和解决问题的能力，对指令和流水段有了更实际地理解。

## 谢北

本次实验课，我在组长的帮助和耐心讲解中，逐步了解并学习了有关CPU的设计，并按照指导教材完成了基本的加减操作和逻辑运算，在实现过程中，我更深一步的认识到了指令的运作过程。但是在整个实验环节中，我独立完成的任务还是少数，希望能在以后的学习和实验过程中，提高自己，在小组作业中贡献出更多的价值。课程安排时间紧任务重，在这种紧锣密鼓的学习节奏中，更能提高我的学习效率和注意力集中程度，实验课老师也很负责任，让我对这次实验课总体感觉非常好。

# 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[M]. v1.01.