成绩评定日期：

实验报告成绩：

2022～2023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、人工智能2202班

组长：沈嘉欣

组员：王香懿

报告日期：2025.1.3

**目录**

**[一、概要 3](#_Toc32021)**

[1.工作量 3](#_Toc15224)

[2.指令集 3](#_Toc31370)

[3.实验环境 4](#_Toc25988)

[4.总体设计 4](#_Toc25988)

[5、流水段连接图 4](#_Toc24723)

[6、实验结果 5](#_Toc20209)

**[二、IF段 5](#_Toc196)**

[1.取指令周期（IF） 5](#_Toc9218)

[2.IF 段设计原理 5](#_Toc22010)

[3.接口定义 6](#_Toc25988)

[4.信号介绍 6](#_Toc13247)

[5. 核心代码 7](#_Toc4987)

**[三、ID段 8](#_Toc19694)**

[1.指令译码/读寄存器周期（ID） 8](#_Toc11981)

[2.ID 段设计原理 8](#_Toc22010)

[3.接口定义 9](#_Toc21445)

[4.信号介绍 9](#_Toc13155)

[5.核心代码 10](#_Toc12472)

**[四、EX段 10](#_Toc11929)**

[1.执行/有效地址计算周期（EX） 10](#_Toc13318)

[2. EX 段设计原理 11](#_Toc17539)

[3.接口定义 11](#_Toc5872)

[4.信号介绍 11](#_Toc8555)

[5.核心代码 12](#_Toc622)

**[五、MEM段 15](#_Toc5221)**

[1.存储器访问／分支完成周期（MEM） 15](#_Toc32040)

[2.MEM 段设计原理 15](#_Toc25023)

[3.接口定义 15](#_Toc897)

[4.信号介绍 16](#_Toc30667)

[5.核心代码 16](#_Toc622)

**[六、WB段 17](#_Toc3951)**

[1.写回周期（WB） 17](#_Toc30559)

[2.WB 段设计原理 18](#_Toc4517)

[3.接口定义 18](#_Toc4032)

[4.信号介绍 18](#_Toc9501)

[5.核心代码 19](#_Toc4776)

**[七、添加指令 19](#_Toc27411)**

[6.1 MIPS 指令格式 19](#_Toc16607)

**[八、心得体会 21](#_Toc7101)**

[1. 沈嘉欣（组长） 21](#_Toc18961)

[2. 王香懿 22](#_Toc18239)

**[九.参考资料 23](#_Toc3837)**

**一、概要**

**1.工作量**

沈嘉欣（占比50%）：负责ID、IF、EX三个基础指令部件编写，负责mul\_plus板块、解决数据相关问题，设计自制乘法器，撰写实验报告。

王香懿（占比50%）：负责MEM、WB两个基础指令部件编写，负责跳转、分支指令的添加撰写实验报告。

**2.指令集**

添加完成了如下指令：

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。 按照写实验的先后循序这些指令顺序如下： 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 成功通过第 64 个点，并制作了一个 32 时钟周期的乘法器。

**3、运行环境及使用工具**

1.使用Verilog语言编写代码。

2.使用Vivado进行仿真。

3.按照实验包中“doc\_v1.0/A03…”中的指令实现一个简易的五级流水结构的CPU。

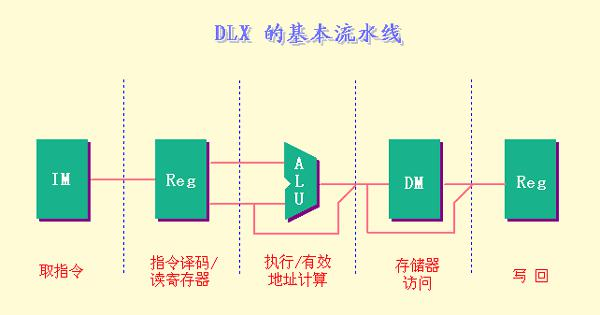
4.操作系统：Windows 10 64 位( DirectX 12 )。

5.开发平台：Vivado 2019.2。

6.编程语言：Verilog HDL 硬件描述语言。

**4、总体设计**

流水线是一种在数字系统中提高系统稳定性和工作速度的方法，被广泛应用于高性能CPU的架构中。根据MIPS处理器的特点，整个处理过程被划分为五个阶段，分别是取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器回写（WB）。这五个阶段对应于多周期的处理，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到来时，代表一条指令的数据和控制信息将被传递到下一个处理阶段。DLX基本流水线的示意图如下：



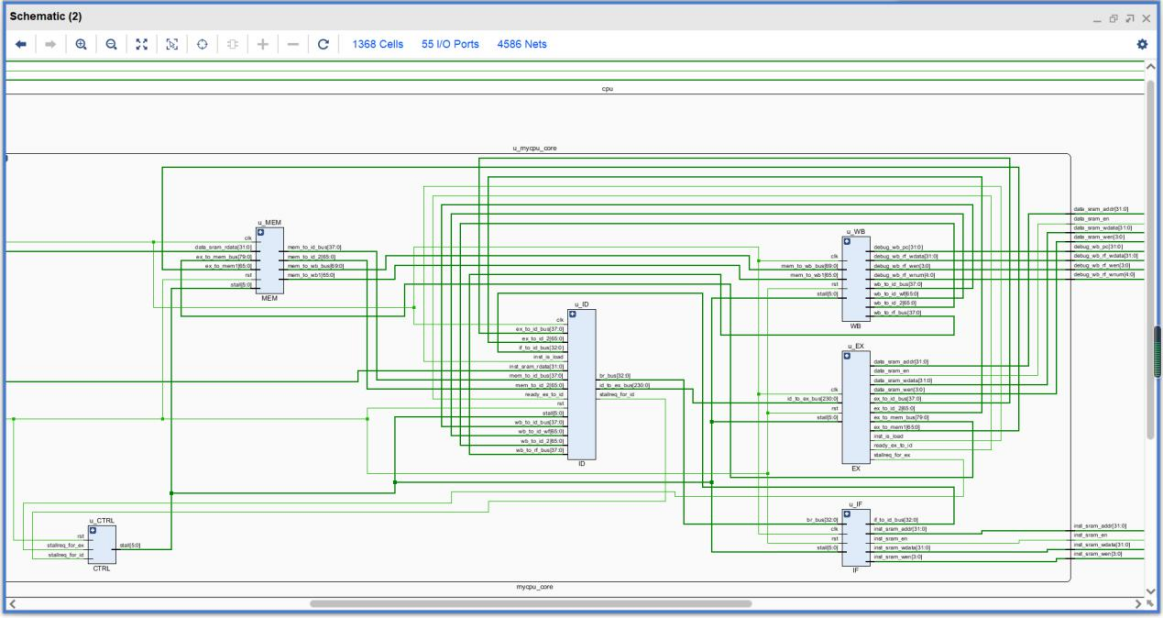
在本方案设计中：

分支指令需要 4 个时钟周期（移到 ID 段，只需 2 个周期）

store 指令需要 4 个时钟周期

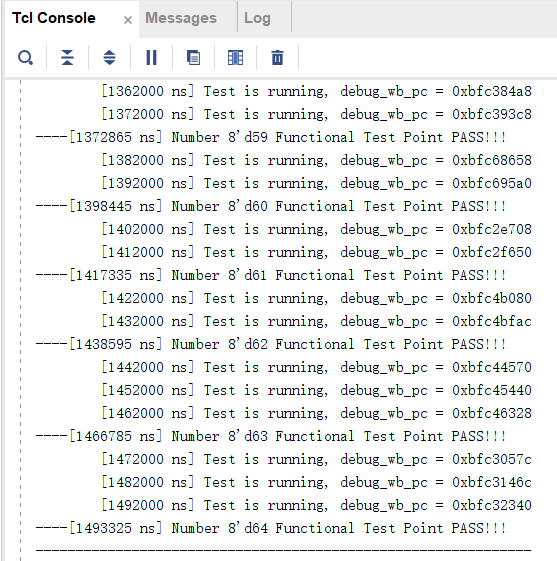
其它指令需要 5 个时钟周期

**5、流水段连接图**



1. **实验结果**

成功通过64个测试点！



**二、IF段**

**1.取指令周期（IF）**

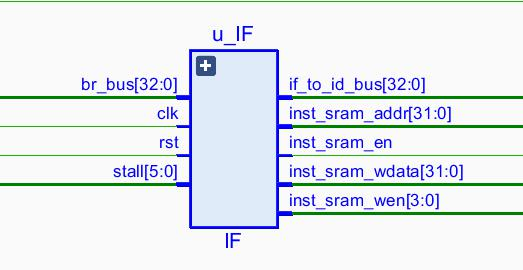
IR ← Mem[PC]

NPC ← PC＋4

该阶段的功能是从指令存储器中提取指令。在此阶段，存在一个PC寄存器，其目的是存储指令的地址。PC输入是由一个选择器选择的结果。从PC寄存器中提取指令地址，根据该地址从指令存储器中提取指令。然后，将提取的指令送入IF/ID流水段寄存器，以便后续流水段使用，并将PC+4的值一并存入流水段寄存器。这一步骤是为了处理分支指令的控制冒险，以便在需要时恢复PC的值。**2. IF 段设计原理**

IF 段的主要职能在于利用程序计数器（PC）中的内容作为地址，从存储器中提取指令，并将其存放到指令寄存器 IR 中。同时，将 PC 的值增加 4（假设每条指令占用 4 个字节），以指向顺序中的下一条指令。IF 段负责根据是否存在跳转条件来为下一个 PC 赋值，并将取指令信号设为真，将总线传递至 WB 段。此外，IF 段将当前计算得到的 PC 值传递给指令存储器，使得在指令译码/读寄存器周期（ID 段）中，指令存储器能够获取当前 PC 值所对应指令的 inst 值。这样，存储器会根据 PC 地址向 ID 段返回一个 32 位的指令（ins 指令），同时将 PC 值传递给 ID 段，以方便后续指令的使用。

**3.接口定义**



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]

**4.信号介绍**

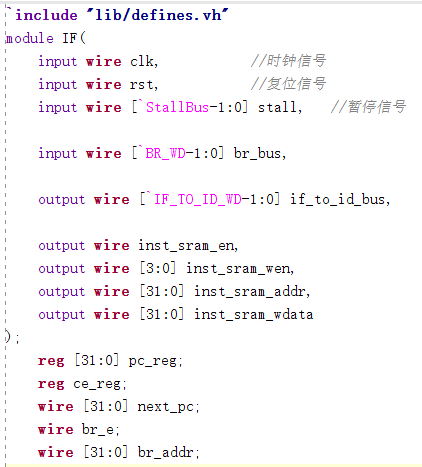
br\_bus[32:0]接收来自ID段的跳转指令，用以改变下一条指令的执行流程。其中包含了br\_e跳转使能信号和br\_addr[31:0]跳转地址值。当br\_e为1且br\_addr[31:0]有值时，将br\_addr[31:0]赋给当前指令的pc值，并发送该pc值至指令寄存器，从而在ID段获取跳转后的指令。

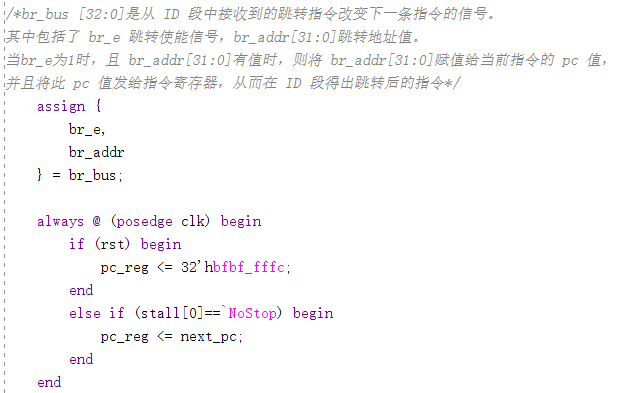
stall[5:0]接收自CTRL.v文件的暂停信号。若stall[0]等于1，则pc值保持前一个时钟周期的数值不变，实现暂停操作。若无暂停信号，则正常执行pc值加4或进行跳转地址操作。

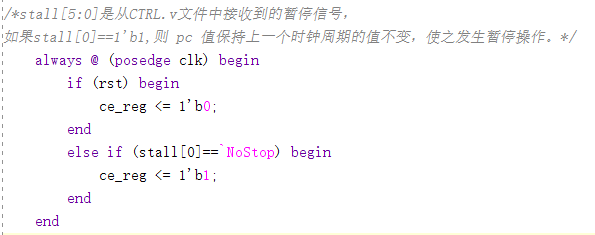
if\_to\_di\_bus[32:0]将当前获取的pc值发送至ID段。inst\_sram\_en是指令存储器读取的使能信号。inst\_sram\_addr[31:0]存储当前pc值，并发送至指令存储器。指令存储器获取pc值后，将其对应的指令inst值发送至ID段，用于译码。

inst\_sram\_wdata[31:0]和inst\_sram\_wen[3:0]用于向指令存储器写入数值。在IF段不涉及存储指令存储器的操作，因此在该段将这两个值都设为0。

1. **核心代码**

****

****

****

**三、ID段**

**1.指令译码/读寄存器周期（ID）**

A ← Regs[IR6 ..10] (Regs[rs])

B ← Regs[IR11 ..15] (Regs[rt])

Imm ← (IR16)16 ## IR16 ..31

该阶段的主要任务是对指令进行解析，执行寄存器访问，并计算跳转指令的目标地址。首先，它将指令进行译码以生成控制信号，并从寄存器组中提取相应的操作数。该阶段从上一个流水段的寄存器中取出指令，并将其送入译码控制器进行译码。一旦译码完成，译码结果将被存入下一个流水段的寄存器。根据译码结果，立即数会经过四种不同的扩展方式，包括逻辑扩展、算术扩展、用于 lui 指令的扩展，以及用于分支指令的扩展。

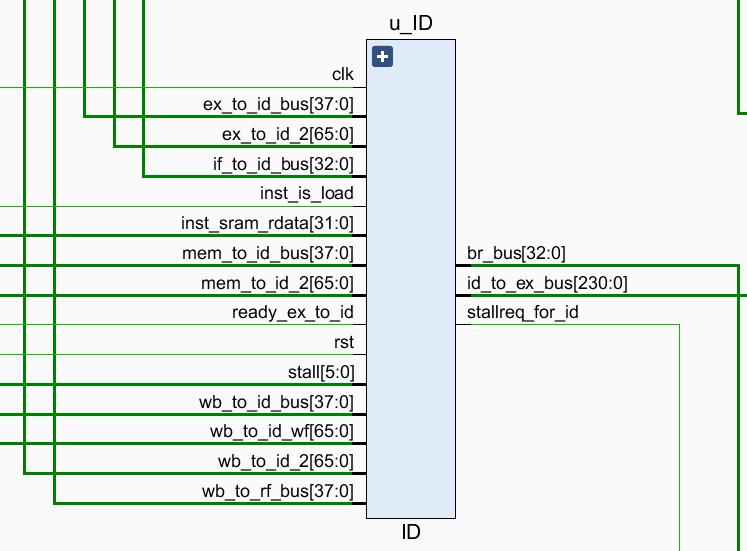
**2.ID 段设计原理**

对于从指令存储器中获取的指令 inst 值，首先进行指令译码。通过使用指令寄存器 IR 中的寄存器地址，访问通用寄存器组，其中包含了 32 个通用寄存器 reg\_array[31:0]。从这些寄存器中读取所需寄存器的值。如果指令中包含立即数，还需要进行符号扩展或无符号扩展。

在处理转移指令时，首先要检查是否满足转移条件。如果条件成立，需要提供转移目标作为新的指令地址。如果解析到的指令是跳转指令，需要计算跳转地址，并通过总线 br.bus 将结果传递回指令获取（IF）段。这确保了在最短的时间内获取下一条指令的地址。

同时，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执行（EX）段。值得注意的是，指令译码和读寄存器是并行进行的，这得益于 DLX 指令格式中操作码在固定位置的设计。这种技术也被称为固定字段译码，使得在同时进行指令解析和寄存器读取的过程中能够高效地处理指令流。

**3.接口定义**



**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

1. **输出端口：**

br\_bus[32:0],id\_to\_ex\_bus[230:0],stallreg\_for\_id

**4.信号介绍**

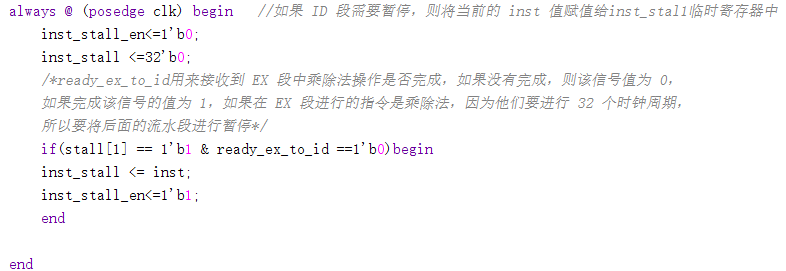
if\_to\_id\_bus[32:0] 从 IF 阶段传输当前处理的 PC 值，其[31:0]部分被分配给 inst 变量。 inst\_sram\_rdata[31:0] 接收当前 PC 对应的指令 inst，ID 阶段在未暂停时会使用这个值进行指令判别，并准备后续指令执行。 ex\_to\_id\_bus[37:0]、mem\_to\_id\_bus[37:0]、wb\_to\_id\_bus[37:0]、ex\_to\_id\_2[65:0]、mem\_to\_id\_2[65:0]、wb\_to\_id\_2[65:0] 是与数据依赖相关的信号，用于处理指令在寄存器值还未更新时的数据需求。这些值由 EX、MEM 和 WB 阶段预先发至 ID 阶段，并通过 ID 阶段传递至 regfile.v，以便为 rs 和 rt 寄存器分配所需的值。

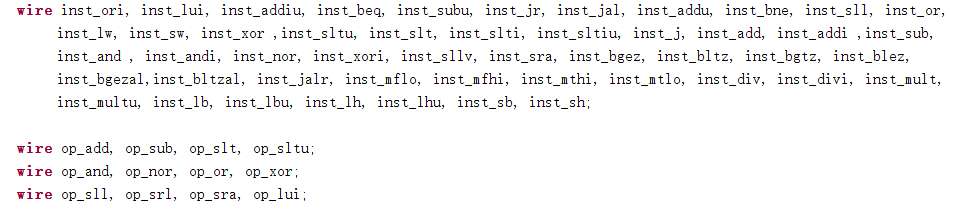
stall[5:0] 从 CTRL.v 接收暂停信号。若 stall[2]==1’b1，则将 if\_to\_id\_bus\_r 清零，保证 IF 阶段的数据不在当前周期内赋值，从而暂停当前周期的操作。同时，将当前周期的 inst 储存在临时寄存器 inst\_stall[31:0] 中，并设置使能信号 inst\_stall\_en 为 1’b1。在下一周期，将这个寄存器的值分配给 inst，待 stall[2]==1’b0 时清空该寄存器，其使能信号 inst\_stall\_en 也重置为 1’b0，实现暂存当前周期 inst 的目的。

ready\_ex\_to\_id 信号从 EX 阶段传来，表明乘除法操作是否完成。若未完成，则信号为 0；完成则为 1。若 EX 阶段的操作为乘除法，需要多个周期完成，ID 阶段将持续保存上一周期的 inst。 wb\_to\_id\_wf[65:0] 和 wb\_to\_rf\_bus[37:0] 从 WB 阶段接收写回寄存器的值，并在 ID 阶段接收，然后传递到 regfile.v 以执行寄存器写操作。 inst\_is\_load 从 EX 阶段接收，用于判断当前指令是否为 LW。若是，则该值为 1；否则为 0。并与 rs 和 rt 寄存器地址比较，若 EX 阶段 LW 指令写入地址与 ID 阶段读取地址相同，则 stallreg\_for\_id 设为 1’b1，并传递给 CTRL.v 以暂停 ID 和 IF 阶段。

br\_bus[32:0] 携带跳转指令目标 PC 值，与跳转使能信号一起发送至 ID 阶段，使下一流水线阶段跳转至目标指令。 id\_to\_ex\_bus[230:0] 是 ID 阶段发送至 EX 的值，包括当前指令的 PC 和 inst 值、alu\_op 操作信号、alu 操作数目标值选择器 sel\_alu\_src1 和 sel\_alu\_src2、访问存储器的使能信号 data\_ram\_en、存储器操作信号 data\_sram\_wen[3:0]、寄存器数组 reg\_array[31:0] 写操作的使能信号 rf\_we、写操作寄存器地址、寄存器 rs 和 rt 存储的值 rdata1 和 rdata2、hilo 寄存器读写操作信号 lo\_hi\_r[1:0] 和 lo\_hi\_w[1:0] 以及对应的 hilo 值 lo\_o 和 hi\_o、对寄存器进行操作的 data\_ram\_read[3:0]。

**5.核心代码**





**四、EX段**

**1.执行/有效地址计算周期（EX）**

存储器访问（load 和 store）

ALUOutput ← A＋Imm

寄存器―寄存器 ALU 操作

ALUOutput ← A op B

寄存器―立即值 ALU 操作

ALUOutput ← A op Imm

分支操作

ALUOutput ← NPC＋Imm 计算偏移地址

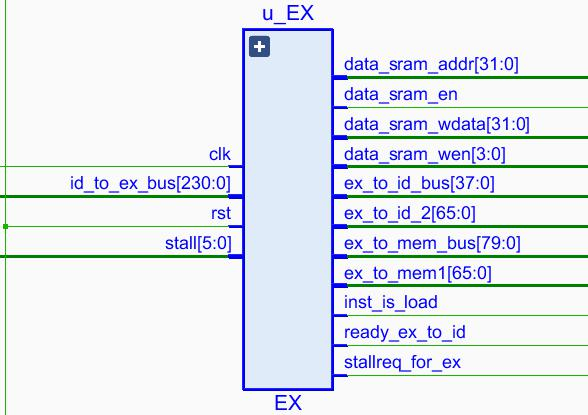
Cond ← (A op 0) 判断分支是否成功，失败则结束

根据指令的编码进行算数或者逻辑运算或者计算条件分支指令的跳转目标地址。 此外 LW、SW 指令所用的 RAM 访问地址也是在本周期上实现。 数据运算是从上一流水段寄存器中取出操作数和控制信号，根据控制信号控制数 据通路进行数据的运算，并将运算的结果存入下一个流水段。但是其取出的操作数也 是进行了转发选择的数据，转发同样还是采用的按字节转发的模式。

**2. EX 段设计原理**

根据解码阶段提供的操作数和运算类型执行运算，并输出结果。如果是load/store指令，还将计算load/store的目标地址。在这个阶段，各种指令会有不同的操作。

**3.接口定义**



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\_ex\_to\_id,stall\_for\_ex

**4.信号介绍**

Id\_to\_ex\_bus[230:0]是传送给EX段的数据，其中包含当前指令的PC值和指令值(inst)，进行ALU操作的信号(alu\_op)，ALU操作数1的目标值选择(sel\_alu\_src1)，ALU操作数2的目标值选择(sel\_alu\_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]，对寄存器文件(reg\_array[31:0])进行写操作的使能信号(rf\_we)，对寄存器文件(reg\_array[31:0])进行写操作的寄存器地址，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]，以及寄存器HILO所对应的数值(lo\_o和hi\_o)，还有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。当stall[3]等于1时，表示需要暂停EX段。特别是当EX段的指令是乘法或除法时，因为这些指令需要32个时钟周期进行计算，所以需要暂停IF、ID和EX段的操作，直到乘除法指令结束后，流水线才能正常运行。

data\_sram\_en是对存储器的访存使能信号。当需要对存储器进行操作时，将其置为1。data\_sram\_addr[31:0]将EX段计算出的结果传送给存储器进行寻址，并将寻址得到的值通过data\_sram\_rdata传递到MEM段。

data\_sram\_wen[3:0]根据当前指令传递到存储器的操作指令，在data\_sram\_wdata[31:0]中控制不同类型要写入存储器的值。

ex\_to\_mem\_bus[79:0]是传送给MEM段的数据，其中包含当前指令的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]，对寄存器文件reg\_array[31:0]进行写操作的使能信号rf\_we，对寄存器文件reg\_array[31:0]进行写操作的寄存器地址rf\_waddr，EX段计算出的结果ex\_result[31:0]，以及对寄存器进行操作的data\_ram\_read[3:0]。

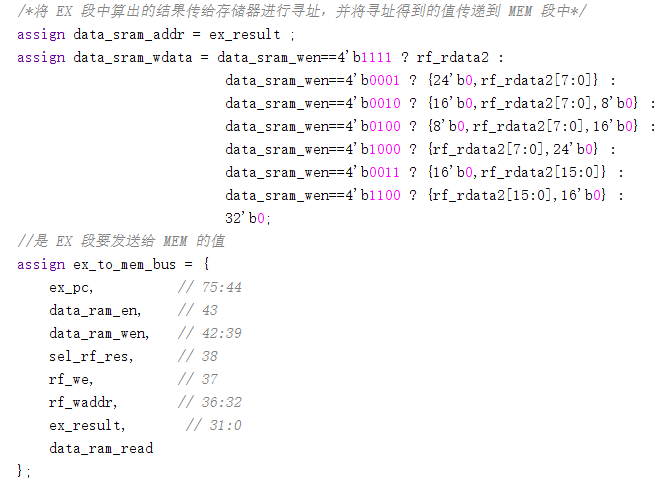
ex\_to\_mem1[65:0]是传送给MEM段的数据，包括写入HI和LO寄存器的使能信号（用于确定是否写入寄存器），还包括要写入HI和LO寄存器的值。如果不进行写入，则值为0且使能信号为0。

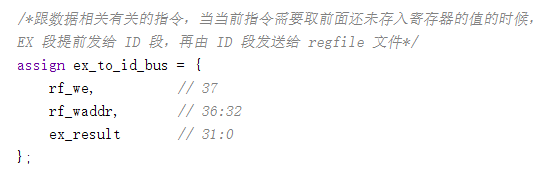
ex\_to\_id\_2[65:0]是传送给ID段中regfile.v的数据，用于解决下一条指令需要使用上一条指令存入HILO寄存器值的问题。

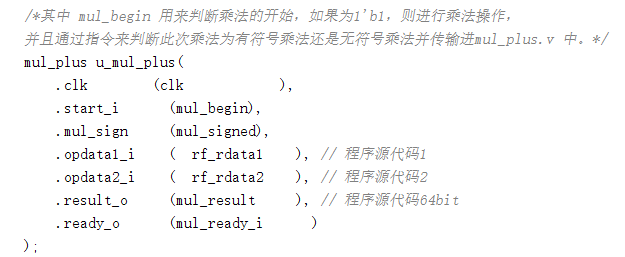
inst\_is\_load是EX段发送到ID段的信号，用于判断EX当前指令是否是LW指令。如果是，则该值为1；如果不是，则该值为0。并且与RS寄存器和RT寄存器中的地址进行比较。如果EX段的LW指令要写入的寄存器地址与当前ID段指令要读取的寄存器地址相同，则stallreg\_for\_id为1，并将此值赋值给CTRL.v，从而暂停ID段和IF段的操作。

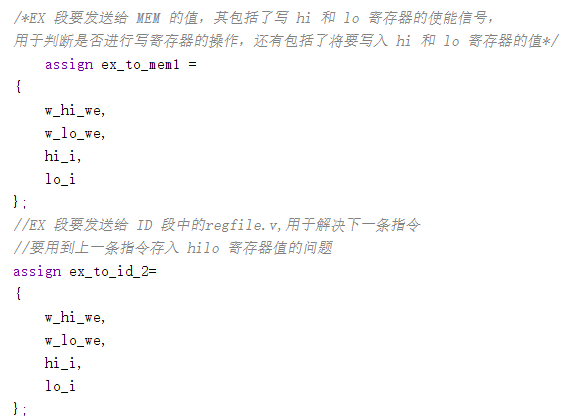
ready\_ex\_to\_id是EX段发送到ID段的一个信号，用于接收EX段中乘除法操作是否完成的信息。如果未完成，则该信号值为0；如果完成，则该信号值为1。如果EX段执行的指令是乘除法，因为需要32个时钟周期，所以需要暂停后续的流水段，因此ID段需要一直保持上一个时钟周期的指令值。

**5.核心代码**

****

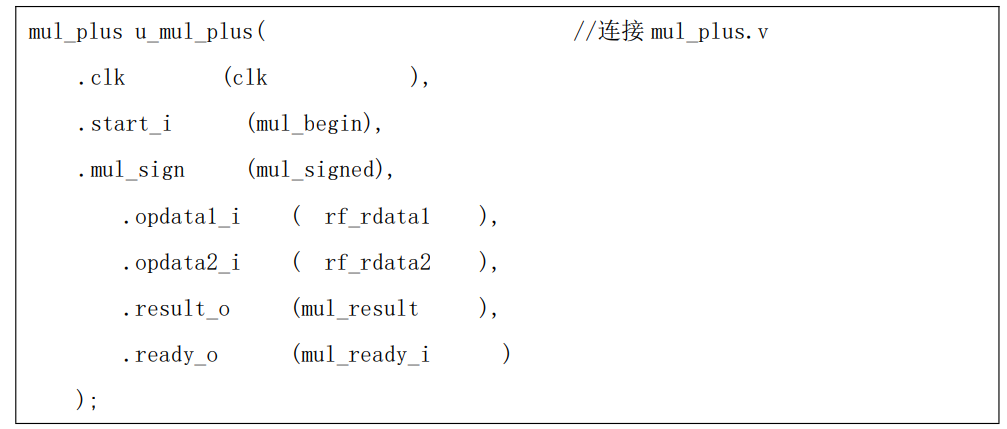
****





**6.自制乘法器mul\_plus说明**

步骤：

首先，对操作数 1 和操作数 2 进行绝对值处理。如果它们的首位为 1 且当前操作是有符号乘法，则将这些操作数取反后再加 1 以获取其绝对值。接下来，操作数 1 的绝对值作为被乘数。被乘数的高 32 位设为 0，低 32 位设为操作数 1 的绝对值；然后将操作数 2 作为乘数的低 32 位。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。同时，乘数每经过一个时钟周期右移一位，最高位补 0，最低位舍弃。在每个时钟周期中，如果乘数的最低位为 1，则将当前时钟周期的被乘数与乘法的临时结果相加，并将结果赋给临时结果；如果最低位为 0，则临时结果保持不变。当乘数不为 0 时，跳回到步骤 3；当乘数为 0 时，结束循环。最后，对于有符号乘法，将操作数 1 和操作数 2 的符号位进行取反或运算，得到最终结果的符号。如果是有符号乘法，将得到的临时结果取反后再加 1；否则，直接将临时结果赋给输出。

**五、MEM段**

**1.存储器访问／分支完成周期（MEM）**

在该周期处理的 DLX 指令只有 Load、Store 和分支指令。 存储器访问（load 和 store）

LMD ← Mem[ALUOutput]

或 Mem [ALUOutput] ← B

分支操作

if（cond）PC ← ALUOutput 成功则把计算好的地址放入 PC

else PC ← NPC 否则不做

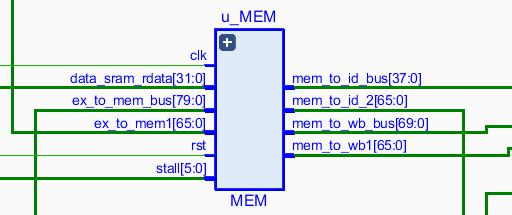
其他类型指令均不做

本阶段进行的操作是内存相关的操作，早内存的读写口之前都有一个移位器，只 是为 LW 和 SW 等类型的指令准备的，还有一个 condition 的检查单元，该单元是用于 判断跳转是否有效和是否真的需要写回寄存器的。

**2.MEM 段设计原理**

如果涉及到 load/store 指令，在这个阶段将会执行数据存储器的访问操作；反之，只需将执行阶段的结果传递到回写阶段。如果存储器传回数值，需要通过相关指令如 lb、lbu、lh、lhu、sb、sh 进行取值操作，并将存储器获取的值进一步存入寄存器。同时，在这个阶段还需要判断是否存在异常需要处理。如果有异常发生，会清除流水线，并转移到异常处理例程入口地址处继续执行。最终，MEM 段会将各种寄存器的读写使能信号、地址和写入数据合并为 MEMTOWB 总线，并将其传递给 WB 段。

**3.接口定义**



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]

**4.信号介绍**

data\_sram\_rdata[31:0] 存储在EX段读取的来自MEM段的数值。

ex\_to\_mem\_bus[79:0] 包含 MEM 段接收的来自 EX 段的值，其中包括当前指令的 pc 值、对存储器执行访存操作的 data\_ram\_en 使能信号、对存储器执行操作的 data\_sram\_wen[3:0]、对寄存器 reg\_array[31:0] 执行写操作的使能信号 rf\_we、对寄存器 reg\_array[31:0] 执行写操作的寄存器地址 rf\_waddr、以及在 EX 段中计算得到的结果 ex\_result[31:0] 和对寄存器执行操作的 data\_ram\_read[3:0]。

ex\_to\_mem1[65:0] 是 MEM 段接收的来自 EX 段的值，其中包括写 hi 和 lo 寄存器的使能信号，用于判断是否执行写寄存器的操作。同时，它还包括将要写入 hi 和 lo 寄存器的值。如果不执行写操作，则这些值为 0，且使能信号为 0。

mem\_to\_id\_2[65:0] 是 MEM 段要发送给 ID 段中的 regfile.v，用于解决下一条指令需要使用上一条指令存入 hilo 寄存器的值的问题。

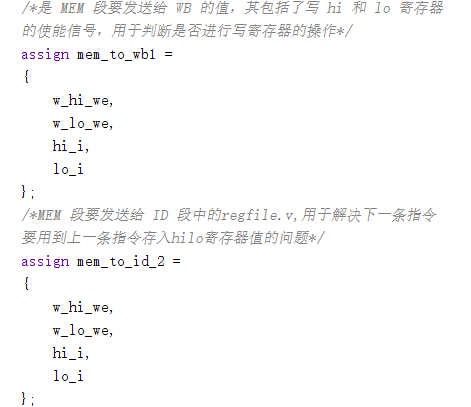
mem\_to\_wb1[65:0] 是 MEM 段要发送给 WB 的值，其中包括写 hi 和 lo 寄存器的使能信号，用于判断是否执行写寄存器的操作。此外，它还包括将要写入 hi 和 lo 寄存器的值。如果不执行写操作，则这些值为 0，且使能信号为 0。

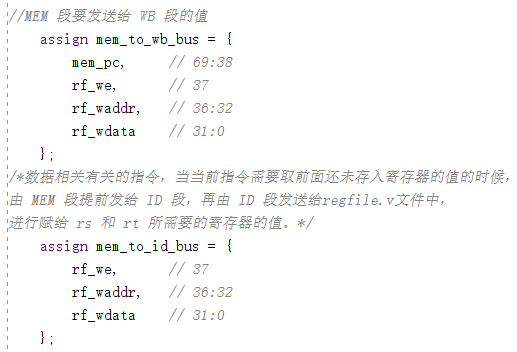
mem\_to\_id\_bus[37:0] 包含与数据相关的指令。当当前指令需要获取尚未存入寄存器的值时，MEM 段会提前将这些值发送给 ID 段，然后由 ID 段发送给 regfile.v 文件，以赋值给 rs 和 rt 所需的寄存器。

mem\_to\_wb\_bus[69:0] 是 MEM 段要发送给 WB 段的值，其中包括当前指令的 pc 值、对寄存器 reg\_array[31:0] 执行写操作的使能信号 rf\_we、对寄存器 reg\_array[31:0] 执行写操作的寄存器地址 rf\_waddr，以及在 EX 段中计算得到的结果 ex\_result[31:0]。

在 MEM 段中，会进行判断，确定最终写入寄存器的值是来自 EX 段传递过来的 ex\_result[31:0] 还是从存储器中传递下来的 data\_sram\_rdata[31:0]。判断后，将相应的值传递给 rf\_wdata。

**5.核心代码**

****



**六、WB段**

**1.写回周期（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

寄存器―立即值型 ALU 指令

Regs[IR11 ..15] (rt)← ALUOutput

Load 指令

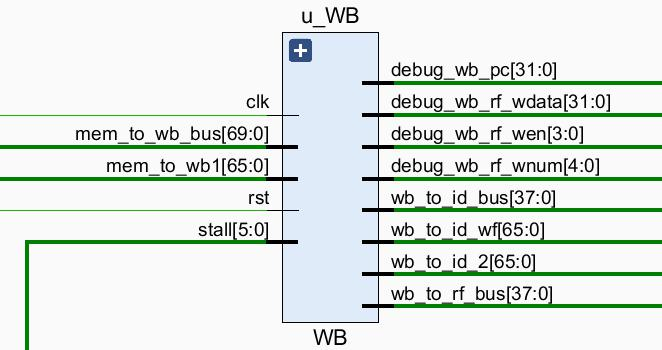
Regs[IR11 ..15] (rt)← LMD

执行的指令将其结果写回寄存器文件，即ALU运算指令和load指令会将结果数据存储到通用寄存器组中。ALU运算指令的结果数据来自ALU，而load指令的结果数据来自存储器。在整个流水线CPU中，时钟关系是按照下降沿对每个流水段寄存器进行写入，并在上升沿进行读取。这是为了处理写回寄存器时可能同时对同一地址进行读写而引起的结构冲突。因此，在设计中采用了读写分离的边沿策略。

**2.WB 段设计原理**

将运算结果保存到目标寄存器。

**3.接口定义**



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]

**4.信号介绍**

rst表示接收到的复位信号，clk是接收到的时钟信号。在这里，mem\_to\_wb\_bus[69:0]代表了WB段从MEM段接收到的值，包括当前指令的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地址rfw\_addr，以及EX段计算得到的结果ex\_result[31:0]。

而mem\_to\_wb1[65:0]是WB段接收到的从MEM段发过来的值，其中包括写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。

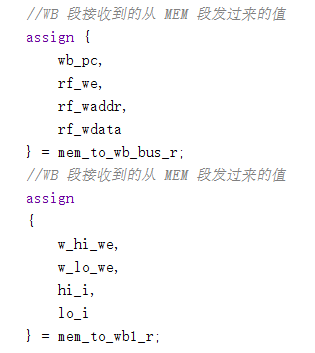
wb\_to\_id\_2[65:0]是WB段要发送给ID段中的regfile.v的值，解决下一条指令需要使用上一条指令存入hilo寄存器值的问题。

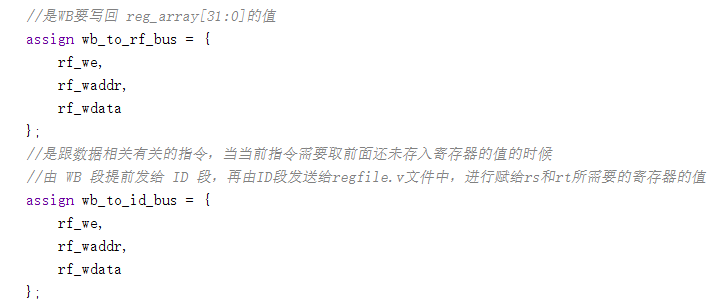
wb\_to\_id\_bus[37:0]包含与数据相关的指令，当当前指令需要获取前面还未存入寄存器的值时，由WB段提前发给ID段，然后由ID段发送给regfile.v文件，以赋给rs和rt所需的寄存器的值。

wb\_to\_rf\_bus[37:0]是WB要写回reg\_array[31:0]的值，包括对寄存器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地址rf\_waddr，以及EX段计算得到的结果ex\_result[31:0]。

最后，wb\_to\_id\_wf[65:0]是WB段要发送给regfile.v的值，包括写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。

**5.核心代码**

****



**七、添加指令**

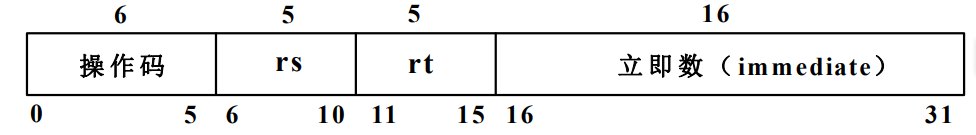
**6.1 MIPS 指令格式**

MIPS 指令系统结构有 MIPS-32 和 MIPS-64 两种结构，本实验的 MIPS 指令选用MIPS-32。以下所说的 MIPS 指令均指 MIPS-32，即 MIPS 的指令格式为 32 位。

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。三类指令格式如下所示。

**1）I 类指令(Immediate)**包括所有的 load 和 store 指令、立即数指令、分支指令、寄存器跳转指令、寄存器链接跳转指令。

立即数字段为 16 位，用于提供立即数或偏移量。



**其中，I 类指令包括：**

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 无用

寄存器跳转、寄存器跳转并链接

转移目标地址为 Regs[rs]

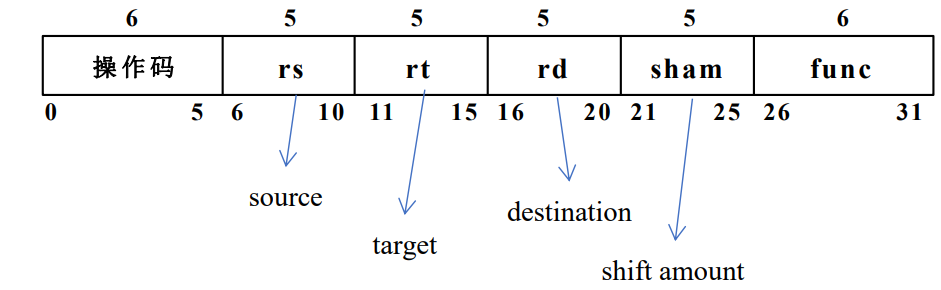
**2）R 类指令(Register-to-Register)**

包括 ALU 指令、专用寄存器读/写指令、move 指令等。

ALU 指令

egs[rd]← Regs[rs] func Regs[rt]

func 为具体的运算操作编码。



**3）J 类指令（Jump）**

包括跳转指令、跳转并链接指令、自陷指令、异常返回指令。

在这类指令中，指令字的低 26 位是偏移量，它与 PC 值相加形成跳转的地址。



处理器实现的指令包括除 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、12 条访存指令，共计 52 条。

**八、心得体会**

1. **沈嘉欣（组长）**

这次计算机系统实验给我的主要感受是入门难度较大。相较于我之前参与的实验课程，这门课程更显得难以掌握。可能是因为它侧重硬件，并且涉及到编写代码的工作，需要我们设计流水线，完成一整套复杂的流水线工作。一开始我感到十分迷茫和焦躁，因为这是第一次真正上手去实现计算机内部的东西，但通过自主学习和复习老师上课讲的知识，我逐渐理解了具体的流程和做法，同时也逐渐开始熟练了vivado的使用和Verilog语言的编写。

在实验中，我时常面临各种困难。在修复自己程序中的错误时，我不仅增强了自己的能力，还学到了许多解决错误的方法。修复自己的bug和他人的bug的过程中，我学到了需要耐心地阅读代码，查找每个变量的定义，并且查看波形图。每个时钟周期都需要仔细查找错误发生的地方。当然，错误的根源并不总是表面上看到的那个位置，这一点在C++代码中也有深刻的体会。因此，调试bug是一个不可避免的问题，需要综合考虑上下文，才能真正找到bug的源头。我相信在调试bug的过程中，我对实验有了更深入的理解，因为经过了一些波折，才能真正学到更多。

我认为在实验过程中，我认为应该先让我们掌握了一些指令后再开始写代码，可能比一味地阅读书籍更为有效。这样可以更好地理解并适应《动手做CPU》中的代码风格，使学习过程更为顺畅。

这次实验让我对计算机系统有了更深入的理解，同时也让我熟悉了vivado工具和Verilog语言的使用。在实验过程中，我遇到了一些挑战，尤其是对指令的添加和参数的理解方面。通过不断学习和实践，我逐渐克服了这些困难，也更加熟练地应用这些知识。此外，团队协作也起到了重要的作用，尤其是在后期进度较为紧张的情况下，大家共同努力，取得了令人满意的成果。

对这次实验的建议，我认为在实验开始前，可以考虑提供更多关于vivado工具和Verilog语言的学习资源，以便学生更迅速地适应和掌握这些工具，有助于提高学生在实验中的学习效果和完成实验任务的效率。

1. **王香懿**

本次计算机系统实验课是用Verilog在Vivado上设计一个CPU。在开始做实验之前，我还不是很理解什么是CPU以及它的工作原理，只知道它是计算机的重要支撑硬件，一系列重要运算都在CPU上进行。完成这个实验课之后，我对CPU的内部组成以及指令在CPU上的运作有了更深一层的了解。

刚拿到题目的时候非常茫然，不知道从何下手，由于之前从来没有接触过Verilog，凭着仅仅在课上学到的知识动手实现一个CPU确实是一个很有挑战的任务。本次实验课的内容相对于之前的C语言、C++、python语言，难度又提升了一个档次，是从未接触过的全新的内容。任务发布下来的时候，我们小组的成员都处于一种云里雾里的状态，感到无从下手，甚至下软件、打开项目都觉得困难重重。虽然我们在老师的课堂上学习了体系结构CPU部件以及相关指令运算，但是我们从来没有实际操作过，在短期内学会一门全新的编程语言Verilog更是难上加难。

但是车到山前必有路，不管再难，只要好好的学习，也一定可以有解决的办法。我和组长分工明确、团结一心，一个bug一个bug的改，仔细研究每一个模块的功能，并且在vivado上通过了仿真测试，很有成就感。本次实验课让我体验到了计算机硬件带给我的全新的体验，也激发了我对CPU的兴趣，加深了我对计算机系统知识的理解。

**九.参考资料**

[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].

[3] 《A03\_“系统能力培养大赛”MIPS 指令系统规范\_v1.01》

[4] 《A07\_vivado 使用说明\_v1.00》

[5] 《A09\_CPU 仿真调试说明\_v1.00》

[6] 《A11\_Trace 比对机制使用说明\_v1.00》