实验 3 报告

第 32 小组
施璠、袁峥

一、实验任务（10%）

1. 设计一款静态 5 级流水简单 MIPS CPU。

2. 本次实验要求延续 lab2 实验中的以下要求：

(1) CPU 复位从虚拟地址 0xbfc00000 处取指。

(2) CPU 虚实地址转换采用：虚即是实。

(3) CPU 对外访存接口为取指、数据访问分开的同步 SRAM 接口。

(4) CPU 只实现一个操作模式：核心模式，不要求实现其他操作模式。

(5) 不要求支持例外和中断。

(6) CPU 顶层连出写回级的 debug 信号，以供验证平台使用。

3. 整个实验中，最后要求实现 MIPS I 指令集，除了 ERET（非 MIPS I）、MTC0、MFC0、BREAK、SYSCALL 指令，其余指令均要求实现，共 56 条指令。

(1) 要求实现 MIPS 架构的延迟槽技术，延迟槽不再设定为 NOP 指令，可能是任意指令。

(2) 控制相关由分支指令造成，通过延迟槽技术可以完美解决。

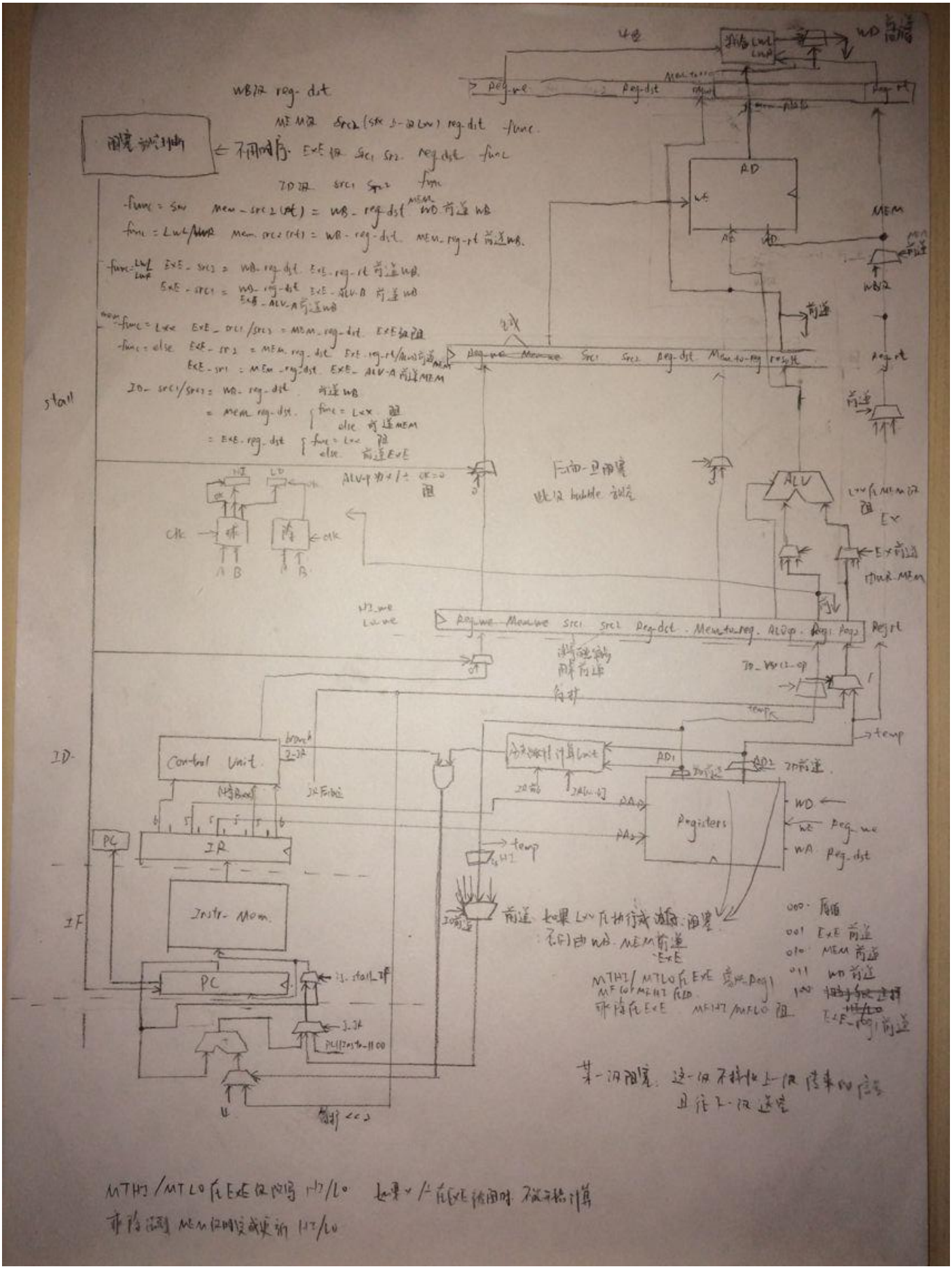
(3) 结构相关即某一级流水停顿了，会阻塞上游的流水级。

(4) 要求数据相关采用前递处理。

(5) 乘除法指令实现可以调用 Xilinx 的乘除法 IP，推荐能力有余的同学自行编写乘除法器，乘法采用 booth 算法+华莱士、除法采用迭代算法。

二、实验设计（30%）

整个 CPU 设计共分为 10 个模块，分别为 ALU、IF_stage、ID_stage、EXE_stage、MEM_stage、WB_stage、next_pc、regfile、stall 和 mycpu。其中 IF_stage、ID_stage、EXE_stage、MEM_stage 和 WB_stage 分别为 CPU 五级流水的阶段，ALU 模块在 EXE_stage 级进行算逻运算，next_pc 产生下条指令地址，regfile 为寄存器堆，stall 为流水线的阻塞总调度，mycpu 是整个设计的顶层，负责其他各个模块的整体调度。整体设计图如下：


Control Unit
Registers
ALU
MEM
IF
ID
EX
stall
Instr. Mem.
PC
IR
Reg_we
Mem_we
Reg_dst
Mem_to_reg
ALUop
WD
WA
RD1
RD2

（一）ALU 模块

（1）基本概述

基本采用了在文档 LEC02_Verilog 复习提供的 alu 代码，修改了其中的几处错误，一处是进行 slt 和 sltu 运算时，减法没有进行运算，另一处是 srl 和 sra 在处理上有些问题，其他地方采用了同样的设计。

（2）接口定义

名称	方向	位宽	功能描述
aluop	IN	6	ALU 操作选择信号
vsrc1	IN	32	ALU 第一个操作数
vsrc2	IN	32	ALU 第二个操作数
result	OUT	32	ALU 运算结果

（二）IF_stage 模块

（1）基本概述

时钟上升沿更新 IF_pc 的值，如果 IF 级没有被阻塞，就用 next_pc，如果 IF 级被阻塞，就进行复位，也就是传递 bubble。同时 IF_inst 记录指令存储器读出的数据。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resten	IN	1	复位信号，低电平有效
IF_stall	IN	1	决定 IF 级是否阻塞
next_pc	IN	32	下一条指令地址
inst_sram_rdata	IN	32	指令存储器的读数据
inst_sram_addr	OUT	32	指令存储器的读地址
IF_pc	OUT	32	IF 级的 pc
IF_inst	OUT	32	IF 级的指令

（三）ID_stage 模块

（1）基本概述

这一级主要是将 IF 级传来的指令存入指令寄存器，并将该条指令进行解码，准备好后面流水级需要的控制信号，同时该级要读取寄存器堆的数据，并完成分支指令是否跳转的判断，然后将结果传回 next_pc。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resten	IN	1	复位信号，低电平有效
ID_stall	IN	1	决定 ID 级是否阻塞
IF_inst	IN	32	IF 级往 ID 级传送的指令
IF_pc	IN	32	IF 级往 ID 级传送的 pc
ID_reg_raddr1	OUT	5	寄存器堆的读地址 1

名称	方向	位宽	功能描述
ID_reg_rdata1	IN	32	寄存器堆的读数据 1
ID_reg_raddr2	OUT	5	寄存器堆的读地址 2
ID_reg_rdata2	IN	32	寄存器堆的读数据 2
ID_br_taken	OUT	1	判断如果 ID 级是分支指令，是否满足跳转条件
ID_br_type	OUT	1	ID 级是否为分支指令
ID_j_type	OUT	1	ID 级是否为跳转指令
ID_jr_type	OUT	1	ID 级是否为寄存器跳转指令
ID_br_index	OUT	16	分支指令的偏移地址
ID_j_index	OUT	26	跳转指令的跳转地址
ID_jr_index	OUT	32	寄存器跳转指令的从寄存器堆读出的跳转地址
ID_vsrc1_for	IN	3	寄存器堆读端口 1 的前递选择信号
ID_vsrc2_for	IN	2	寄存器堆读端口 2 的前递选择信号
HI	IN	32	前递传来的 HI 寄存器的值
LO	IN	32	前递传来的 LO 寄存器的值
EXE_for	IN	32	从 EXE 级前递传来的值
MEM_for	IN	32	从 MEM 级前递传来的值
WB_for	IN	32	从 WB 级前递传来的值
ID_pc_out	OUT	32	ID 级往 EXE 级传递的 pc
ID_inst_out	OUT	32	ID 级往 EXE 级传递的指令
ID_src1_out	OUT	5	ID 级指令读寄存器堆时的读地址 1
ID_src2_out	OUT	5	ID 级指令读寄存器堆时的读地址 2
ID_dest_out	OUT	5	ID 级指令写回时的目的寄存器
ID_ALUop_out	OUT	6	ID 级指令 ALU 操作数
ID_vsrc1_out	OUT	32	ID 级指令读寄存器堆的读数据 1
ID_vsrc2_out	OUT	32	ID 级指令读寄存器堆的读数据 1
ID_reg_rt_out	OUT	32	ID 级指令 rt 寄存器的读数据（针对 LOAD 指令）
ID_HI_we_out	OUT	1	ID 级指令 HI 寄存器的写使能信号
ID_LO_we_out	OUT	1	ID 级指令 LO 寄存器的写使能信号
ID_inst	OUT	32	ID 级的指令

（四）EXE_stage 模块

（1）基本概述

这一级主要是将 ID 级传来的两个数据进行 ALU 运算，并将结果送给 MEM 级，同时如果是进行的乘除法操作，需要多拍才能完成运算，因此需要将后续指令进行阻塞。乘除法完成后，将运算结果送到 HI 和 LO 寄存器。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resten	IN	1	复位信号，低电平有效
EXE_stall	IN	1	决定 EXE 级是否阻塞
ID_pc	IN	32	ID 级往 EXE 级传送的指令
ID_inst	IN	32	ID 级往 EXE 级传送的 pc
ID_src1	IN	5	ID 级指令的读寄存器地址 1
ID_src2	IN	5	ID 级指令的读寄存器地址 2
ID_dest	IN	5	ID 级指令的写回寄存器地址
ID_ALUop	IN	6	ID 级指令的 ALU 操作选择信号
ID_vsrc1	IN	32	ID 级传递的 ALU 操作数 1
ID_vsrc2	IN	32	ID 级传递的 ALU 操作数 2
ID_reg_rt	IN	32	ID 级传递的 rt 寄存器的读数据（针对 LOAD 指令）
ID_HI_we	IN	1	ID 级传递的 HI 寄存器写使能信号
ID_LO_we	IN	1	ID 级传递的 LO 寄存器写使能信号

名称	方向	位宽	功能描述
EXE_srcA_for	IN	2	EXE 级的 ALU 操作数 1 的前递选择信号
EXE_srcB_for	IN	2	EXE 级的 ALU 操作数 2 的前递选择信号
EXE_reg_rt_for	IN	2	EXE 级指令的 rt 寄存器的前递选择信号
MEM_for	IN	32	从 MEM 级前递传来的值
WB_for	IN	32	从 WB 级前递传来的值
EXE_pc_out	OUT	32	EXE 级的 pc
EXE_inst_out	OUT	32	EXE 级的指令
EXE_src1_out	OUT	5	EXE 级指令的读寄存器地址 1
EXE_src2_out	OUT	5	EXE 级指令的读寄存器地址 2
EXE_dest_out	OUT	5	EXE 级指令的写会寄存器地址
EXE_result_out	OUT	32	EXE 级 ALU 运算结果
EXE_reg_rt_out	OUT	32	EXE 级指令 rt 寄存器的读数据（针对 LOAD 指令）
EXE_HI_we_out	OUT	1	EXE 级指令的 HI 寄存器的写使能信号
EXE_LO_we_out	OUT	1	EXE 级指令的 LO 寄存器的写使能信号
EXE_for	OUT	32	EXE 级向前传送的前递信号
EXE_inst	OUT	32	EXE 级指令
EXE_dest	OUT	5	EXE 级指令的回写寄存器
EXE_memaddr	OUT	32	EXE 级提前送给 MEM 级的访存读地址
EXE_data_sram_ren	OUT	1	EXE 级提前送给 MEM 级的访存读使能信号
HI_out	OUT	32	EXE 级乘除法指令运算结果
LO_out	OUT	32	EXE 级乘除法指令运算结果
EXE_mul_div_validout	OUT	1	EXE 级乘除法指令输出有效信号

（五）MEM_stage 模块

（1）基本概述

这一级完成数据存储器的读写操作，如果是 LOAD 指令，读地址在 EXE 级已经提前送到数据存储器的地址端口，等到 MEM 级上升沿到来，即完成读取数据，然后将其存入寄存器。如果是 STORE 指令，在当拍计算出写使能信号，并传递数据存储器的写数据和写地址，在下一拍上升沿时完成数据存储器的更新。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resten	IN	1	复位信号，低电平有效
EXE_pc	IN	32	EXE 级的 pc
EXE_inst	IN	32	EXE 级的指令
EXE_src1	IN	5	EXE 级指令的读寄存器地址 1
EXE_src2	IN	5	EXE 级指令的读寄存器地址 2
EXE_dest	IN	5	EXE 级指令的寄存器写回地址
EXE_result	IN	32	EXE 级 ALU 运算结果
EXE_reg_rt	IN	32	EXE 级传递的 rt 寄存器数据（LOAD 指令在 WB 级使用）
EXE_memaddr	IN	32	EXE 级提前准备的访存读地址
EXE_data_sram_ren	IN	1	EXE 级提前准备的访存读使能信号
MEM_reg_rt_for	IN	1	MEM 级 rt 寄存器传递数据的前递选择信号
WB_for	IN	32	从 WB 级前递传来的值
data_sram_rdata	IN	32	数据存储器的读数据
data_sram_en	OUT	1	数据存储器的片选信号
data_sram_wen	OUT	4	数据存储器的写使能信号
data_sram_wdata	OUT	32	数据存储器的写数据
data_sram_addr	OUT	32	数据存储器的地址

名称	方向	位宽	功能描述
MEM_pc_out	OUT	32	MEM 级的 pc
MEM_inst_out	OUT	32	MEM 级的指令
MEM_reg_we_out	OUT	2	MEM 级写地址的最后两位（用于 LOAD 指令在 WB 级拼接写回寄存器的值）
MEM_dest_out	OUT	5	MEM 级指令的写回寄存器地址
MEM_memtoreg_out	OUT	1	MEM 级指令写回寄存器数据选择信号
MEM_result_out	OUT	32	MEM 级传递的 ALU 计算结果
MEM_mem_rdata_out	OUT	32	MEM 级数据存储器的读数据
MEM_reg_rt_out	OUT	32	MEM 级传递的 rt 寄存器数据（LOAD 指令在 WB 级使用）
MEM_for	OUT	32	MEM 级向前传送的前递信号
MEM_src2	OUT	5	MEM 级指令读寄存器的地址 2

（六）WB_stage 模块

（1）基本概述

将 ALU 模块的计算结果或者数据存储器读取的结果送到寄存器堆改写相应寄存器。这里需要注意的情况是如果是 LOAD 指令，需要写回的结果与原寄存器里的数据可能有关，因此将原寄存器中的数据用 MEM_reg_rt 一路送到 WB 级，并根据访存地址的最后两位（即 MEM_reg_we 信号），在 WB 级进行拼接后再送到寄存器堆进行改写。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
resten	IN	1	复位信号，低电平有效
MEM_pc	IN	32	MEM 级传递的 pc
MEM_inst	IN	32	MEM 级传递的指令
MEM_reg_we	IN	2	MEM 级传递的访存地址最后两位，用于与 MEM_reg_rt 信号结合，进行 LOAD 指令的写回数据拼接
MEM_dest	IN	5	MEM 级传递的写回寄存器号
MEM_memtoreg	IN	1	MEM 级传递的写回数据选择信号
MEM_result	IN	32	MEM 级传递的 ALU 运算结果
MEM_mem_rdata	IN	32	MEM 级访存的读数据
MEM_reg_rt	IN	32	MEM 级传递的 rt 寄存器数据（用于 LOAD 指令）
WB_reg_write_en	OUT	1	WB 级写回写使能信号
WB_reg_wdata	OUT	32	WB 级写回数据
WB_reg_addr	OUT	5	WB 级指令写回寄存器号
WB_pc	OUT	32	WB 级的 pc
WB_for	OUT	32	WB 级向前传送的前递信号

（七）next_pc 模块

（1）基本概述

根据 ID 级送出的分支跳转相关信号，计算下一条指令地址，并提前送到 IF 级的指令存储器读地址。

（2）接口定义

名称	方向	位宽	功能描述
IF_pc	IN	32	IF 级的 pc
ID_br_taken	IN	1	ID 级分值跳转是否成功
ID_br_type	IN	1	ID 级指令是否为分支指令

名称	方向	位宽	功能描述
ID_j_type	IN	1	ID 级指令是否为跳转指令
ID_jr_type	IN	1	ID 级指令是否为寄存器跳转指令
ID_br_index	IN	16	ID 级分支指令偏移地址
ID_j_index	IN	26	ID 级跳转指令跳转地址
ID_jr_index	IN	32	ID 级寄存器跳转指令跳转地址
inst_sram_en	OUT	1	指令存储器读使能信号
next_pc	OUT	32	下一条指令地址

（八）regfile 模块

（1）基本概述

该寄存器堆为异步读同步写的，在 WB 级写寄存器堆，在后一上升沿完成更新；在 ID 级读寄存器堆。

（2）接口定义

名称	方向	位宽	功能描述
clk	IN	1	时钟信号
raddr1	IN	5	寄存器堆读地址 1
rdata1	OUT	32	寄存器堆读数据 1
raddr2	IN	5	寄存器堆读地址 2
rdata2	OUT	32	寄存器堆读数据 2
we	IN	1	寄存器堆写使能信号，高低平有效
waddr	IN	5	寄存器堆写地址
wdata	IN	32	寄存器堆写数据

（九）stall 模块

（1）基本概述

该模块整体控制整个五级流水的阻塞或运行，并提供不同端口的前递选择信号。

整个设计中需要用到前递选择的地方一共有 6 处，分别是 ID 级中两个寄存器读端口读出的数据，EXE 级 ALU 模块两个输入口的数据，以及 EXE 级和 MEM 级传递的 reg_rt 信号（该信号是用于 LOAD 指令在 WB 级与原 rt 寄存器值进行拼接）。

五级流水中，可能会出现阻塞的有 ID 级、IF 级和 EXE 级。出现阻塞的情况如下：

1、 MEM 级中是 LOAD 指令，EXE 级指令的寄存器读地址和 MEM 级寄存器写回地址相同时，EXE 级及前面流水阻塞等待。

2、 MEM 级是 STORE 指令且 EXE 级是 LOAD 指令，EXE 级及前面流水阻塞等待，因为 LOAD 指令的访存地址需要提前送到 MEM 级的数据存储器的地址端口，但此时 STORE 指令还在用该端口写数据存储器，因此 LOAD 指令需要阻塞等待。

3、 EXE 级是乘除法指令，且乘除法结果有效信号为 0，即还没有完成运算，EXE 级及前面的流水阻塞等待。

4、 MEM 级中是 LOAD 指令，ID 级指令的寄存器读地址和 MEM 级寄存器写回地址相同时，ID 级及前面流水阻塞

等待。

5、EXE 级中是 LOAD 指令，ID 级指令的寄存器读地址和 EXE 级寄存器写回地址相同时，ID 级及前面流水阻塞等待。

6、EXE 级是乘除法指令，ID 级是 MFHI/MFLO 指令，即使乘除法完成运算，ID 级和 IF 级仍需要阻塞等待，因为 HI 和 LO 寄存器需要等到下一拍上升沿才能完成更新。

（2）接口定义

名称	方向	位宽	功能描述
WB_reg_dst	IN	5	WB 级指令写回寄存器号
MEM_src2	IN	5	MEM 级指令读寄存器地址 2（用于 LOAD 指令）
MEM_reg_dst	IN	5	MEM 级指令写回寄存器号
MEM_inst	IN	32	MEM 级指令
EXE_src1	IN	5	EXE 级指令读寄存器地址 1
EXE_src2	IN	5	EXE 级指令读寄存器地址 2
EXE_reg_dst	IN	5	EXE 级指令写回寄存器号
EXE_inst	IN	32	EXE 级指令
ID_src1	IN	5	ID 级指令读寄存器地址 1
ID_src2	IN	5	ID 级指令读寄存器地址 2
ID_inst	IN	32	ID 级指令
EXE_mul_div_validout	IN	1	EXE 级乘除法结果有效信号
IF_stall	OUT	1	IF 级是否阻塞
ID_stall	OUT	1	ID 级是否阻塞
EXE_stall	OUT	1	EXE 级是否阻塞
ID_vsrc1_for	OUT	3	ID 级读寄存器输出口 1 前递选择信号
ID_vsrc2_for	OUT	2	ID 级读寄存器输出口 2 前递选择信号
EXE_srcA_for	OUT	2	EXE 级 ALU 输入口 1 前递选择信号
EXE_srcB_for	OUT	2	EXE 级 ALU 输入口 2 前递选择信号
EXE_reg_rt_for	OUT	2	EXE 级 reg_rt 信号前递选择信号
MEM_reg_rt_for	OUT	1	MEM 级 reg_rt 信号前递选择信号

（十）mycpu 模块

该模块是整个设计的顶层，进行不同模块之间的连接，同时存有 HI 和 LO 寄存器，根据 EXE 级传递的信号（乘除法指令完成信号）在时钟上升沿进行更新。

三、实验过程（60%）

（一）实验流水账

9 月 30 日下午 2 点到 5 点，阅读文档 A05_“体系结构研讨课”MIPS 指令系统规范、A07_交叉编译工具链安装、LEC02_Verilog 复习、LEC03_CPU 实验开发环境使用说明、LEC04_仿真调试说明和 Lab03_静态 3 级流水简单 MIPS CPU 实现。

9 月 30 日晚上 7 点到 11 点，阅读文档 REF01_计算机体系结构_第五章_静态流水线，并开始绘制设计图。

10 月 1 日上午 10 点到晚上 11 点，绘制设计图。

10 月 2 日下午 2 点到晚上 11 点，编写代码。

10 月 3 日上午 10 点到 12 点，编写代码。

10 月 3 日下午 2 点到晚上 11 点，调试程序，并通过仿真。

10 月 4 日下午 2 点到晚上 11 点，处理乘除法部分，编写除法器程序。

10 月 9 日下午 3 点到晚上 11 点，发现前递部分在综合时时序有问题，进行修改调试，并通过上板测试。

（二）错误记录

1、错误 1

（1）错误现象

在仿真时发现波形上出现一些 X 或者 Z 信号。

（2）分析定位过程

在波形图中和程序中找到出现 X 信号的位置，并结合具体代码确定该 X 信号是由哪个信号引起的，然后再这样一路往前追踪，直到找到源头。

（3）错误原因

X 信号的出现是由于在多路选择时可能某一路信号没有准备好，或者定义的信号位数前后不一致。

Z 信号的出现是由于在顶层调度时某一接口的信号漏接。

（4）修正效果

根据设计的需要和真正的想法修改信号的连接，成功解决该类问题。

（5）归纳总结

解决此类问题主要是要求在编写程序时更加细心，特别在顶层接线要仔细，不要写错变量名和信号长度等。

2、错误 2

（1）错误现象

ALU 中 slt、sltu、sra 和 srl 出错。

（2）分析定位过程

跟踪到 EXE 级中的 ALU 模块信号，结合输入数据并手动计算，发现计算结果错误。

（3）错误原因

之前直接采用了文档 LEC02_Verilog 复习提供的 alu 代码，里面在这几条指令的处理上有错误。

（4）修正效果

重新分析了计算过程，并在减法选择信号中加入了 slt、sltu 信号，并重写了 sra 和 srl 指令的操作。

（5）归纳总结

不要偷懒，复用别的代码前还需要自己先分析和仔细检查一遍。

3、错误 3

（1）错误现象

仿真通过后在综合和布局布线时报错，报的错误有 time loop 和 time violation。

（2）分析定位过程

一开始看到 time loop 的 critical warning 时很费解，反复检查了上下文发现并没有什么明显的错误，然后进一步检查了布局布线的时序报告，发现不能满足时序要求，接着查找到了超时路径的信息，发现有一些信号从数据存储器一路连到了指令存储器。在与邢老师的交流下，重新对数据前递有了一些理解，某一拍计算出来的结果必须先存到寄存器然后才能前递，因此修改了设计图。

（3）错误原因

错误原因主要是对数据前递的理解不够深刻，导致在时序上没有注意，从而使得从数据存储器读出数据直接被送到了指令存储器，该路径超过所要求的流水级时间。

（4）修正效果

根据要求重新修改了设计图，原来在读数据存储器时没有提前给地址，而是将读出的数据直接送到回写级，因此该数据没有经过寄存器保存，如果同时将该数据进行前递那么就会出现路径过长的状况。因此此处修改为提前送指令存储器的读地址，然后将读出的数据存入寄存器，从而在下一拍再进行前递，这样就可以解决这个问题。

（5）归纳总结

没有对数据前递进行深刻理解，在以后必须充分了解了以后再进行设计和编写代码，同时在设计是不能够只管仿真的结果，也应该注意综合和布局布线是否合理。

4、错误 4

（1）错误现象

访问指令 ram 的时候，得不到应有的指令。

（2）分析定位过程

根据波形，发现在取值阶段 inst_sram_en 被置为 0.

（3）错误原因

对 de_br_taken 信号的错误理解，并写成了只有在只有在 de_br_taken 有效时，可以访问指令 ram。

（4）修正效果

修改时把 inst_sram_en 恒置为 1，保证指令 ram 一直可以读，通过后面的控制信号决定要不要更新 nextpc

（5）归纳总结

同步 ram 和异步 ram 相比延迟了一周产生结果，在取值阶段的实际中，需要提前取出 nextpc，那么在真正的取值阶段就可以得到当前所需的指令。 同样地，在 br 操作中需要在译码阶段就判断是否需要跳转。

5、错误 5

（1）错误现象

仿真阶段 jal 指令发生错误

（2）分析定位过程

本来应该写回 pc+8, 但是 mycpu 没有写回值。通过回想 mips 指令集，发现在这条指令上理解和设计发生偏差。

（3）错误原因

在译码阶段没有给 jal 指令设置控制信号，错误地认为在此阶段产生信号控制 nextpc 而后面阶段不需要再执行。所以在运行过程中并没有把 pc+8 写回 31 号寄存器。考虑到后面地指令执行已经根据译码级设计好的 de_op 传递的信号进行控制，对信号进行修改工作量较大，所以直接给新增了一位控制信号用来判断 jal 指令。相应地需要修改 jal 指令在执行阶段的两个操作数和 alu 控制信号。

（4）修正效果

按上述方法修改 jal 指令的相关操作后，可以得到正确结果。

（5）归纳总结

设计之初要对指令有深入了解，不要盲目动手写代码，要先设计好，否则修改起来很麻烦。

6、错误 6

（1）错误现象

LW 指令执行后，load 写回的值不是 ram 中的值，而是 alu 计算后的数据

（2）分析定位过程

在设计中，控制 load 的写入选择的信号，是在译码阶段产生的 op 逐级传递的。根据波形显示，此信号将 alu 的计算结果写会到了寄存器中，也就是此处的信号错了。追踪波形，发现此处的控制信号已经是下一条指令的信号，而不是当前指令的信号。考虑到在访存阶段对输出 mem_value 使用了时序逻辑，而控制信号也是在此阶段由时序逻辑产生，从而找到了错误的地方。

（3）错误原因

访存阶段对 mem_value 的赋值过程如下，当时的设计为了保持住 lw 的值，在写回过程中，如果是 lw 操作，就直接把此阶段的 mem_value 传给 wb_rf_wdata，也就是写回数据。在 mem_value 的赋值中产生了竞争冒险，在时钟上升沿到来时，mem_op 可能是旧值，也可能是此阶段的新值，从而导致了信号的错误。

```
79  always @ ( posedge clk) begin
80    if(!resetn) mem_value <= 0;
81    else if(mem_op[4])  mem_value <= data_sram_rdata; //lw
82    else mem_value <= exe_value;  //other
83  end
```

（4）修正效果

修改方法是把 mem_value 理解成上一拍中 exe_value 传递下来的值，而对 wb_rf_wdata 的选择放到写回操作中执行，设置一个寄存器，保存访存阶段中 data_sram_rdata 的值，并把它传递给写回周期。

访存周期中的修改：

```
80   always @ ( posedge clk) begin
81      if(!resetn) mem_value <= 0;
82      else mem_value <= exe_value;
83   end
84   always @ (posedge clk) begin
85     data_sram_rdata_out <= data_sram_rdata;
86   end
```

写回周期中的修改：

```
93   assign wb_rf_wdata = (wb_op[4]) ? data_sram_rdata_out : wb_value;
```

这样修改后，保证了 wb_rf_wdata 是根据指令控制信号选择的正确的值。

（5）归纳总结

在设计时要综合考虑，避免竞争冒险，对信号的产生和使用要有清晰的理解和认识，然后再动手写代码。

四、实验总结

（一）组员：袁峥

这次实验总体来说还算顺利，就是花费的时间有点多，先花了一天阅读了相关的文档，然后花了一天半画完了设计图，花了一天编写程序，花了半天进行调试。目前整个实验的完成情况是还缺少自己编写的乘法器模块，其他内容应该都已经全部完成。在进行整体设计的时候，主要碰到的问题有以下几处：

1、在前递部分花了很长时间思考，具体哪些地方需要前递，以及前递信号如何准备，这方便考虑起来比较麻烦而且容易出错。最后参考了文档 REF01_计算机体系结构_第五章_静态流水线中的处理办法，根据实际情况进行

了调度。最后在综合部分还是出现了问题，前递前没有把数据存入寄存器，导致路径过长。在与老师交流后认知到了错误并进行了改正。

2、一开始在设计时思考，如果延迟槽指令也是分支跳转指令怎么办，并且百思不得其解，后来在 MIPS 手册中看到，规范中保证了这种情况不会发生，也就解决了这个问题。

3、乘除法器操作如果放在一拍内完成会导致 EXE 级过长，因此需要切分流水，但是具体需要切分多少拍不清楚。在咨询老师后决定乘法器 2 拍完成，除法器可以用 16 拍或者 32 拍完成。乘法器目前还没有完成编写，除法器使用原码加减交替法完成，具体内容在下次实验报告中阐述。

4、整个流水控制中怎么进行阻塞。一开始准备在每两级流水之间相互联系，但后来发现这种方式可能过于繁琐，因此新加了一个 stall 模块来整体控制流水级的阻塞，而被阻塞的流水级往下一级传递的信号全部为 0，本级流水寄存器在下一拍时钟上升沿时也不进行更新，仍然保存旧值，这样也就完成了阻塞的操作。

5、LWR、LWL、SWL、SWR 这四条指令的理解上花了很长时间，反复查看了 MIPS 手册，并且结合计算机体系结构课的课件和配图，最后理解了具体操作。

6、LOAD 指令在写回寄存器时可能会需要与指令中的 rt 寄存器原来的值进行拼接，因此在整个流水中需要将 rt 寄存器的值一直传递到 WB 级，其中也会需要用到前递，并在 WB 级与从数据存储器中读取的数据进行拼接，然后再送到寄存器堆的写数据完成数据更新。

（二）组员：施璠

这次实验是和队友分开做的，在队友差不多把后面几个阶段都写完了，我就完全没有压力地开始了第一个阶段。考虑到之前的多周期实验的写法有点乱，本次实验没有在之前的多周期基础上改，而是根据流水分模块重新写。本来想这一阶段把所有的指令都实现的，但是写到后面发现有一些要处理的问题还挺复杂的，虽然译码级把所有指令都包含了，但是执行级访存级和写回级都只考虑了第一个阶段要实现地指令。在本次实验中，学习了有关五级流水的知识，也反应了一些问题，比如开始写实验之前没有完整的设计思路，想到哪里写到哪里。结果就是写了一天 bug，又改了一天 bug。在 debug 的过程中，不断发现此前考虑的漏洞，补救起来就有点麻烦。希望以后想通了再开始写。