浙江大学

本科实验报告

课程名称：	计算机组成与设计实验
姓　　名：	李依林
学　　院：	信息与电子工程学院
专　　业：	信息工程
学　　号：	3170101212
指导教师：	屈民军、唐奕

2019 年 12 月 23 日

专业：信息工程
姓名：李依林
学号：3170101212
日期：2019.12.23
地点：玉泉教 11

浙江大学实验报告

课程名称：计算机组成与设计实验　指导老师：屈民军、唐奕　成绩：_____
实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计

一、实验目的

（1） 了解 RISC-V 指令系统。
（2） 了解提高 CPU 性能的方法。
（3） 掌握流水线 RISC-V 微处理器的工作原理。
（4） 理解数据冒险、控制冒险的概念以及流水线冲突的解决方法。
（5） 掌握流水线 RISC-V 微处理器的测试方法。
（6） 了解用软件实现数字系统的方法。

二、实验内容

(一）基本要求

设计一个流水线 RISC-V 微处理器，具体要求如下所述。

（1）至少运行下列 RV32I 核心指令。

① 算术运算指令：add、sub、addi
② 逻辑运算指令：and、or、xor、slt、sltu、andi、ori、xori、slti、sltiu
③ 移位指令：sll、srl、sra、slli、srli、srai
④ 条件分支指令：beq、bne、blt、bge、bltu、bgeu
⑤ 无条件跳转指令：jal、jalr
⑥ 数据传送指令：lw、sw、lui、auipc
⑦ 空指令：nop

（2）采用 5 级流水线技术，对数据冒险实现转发或阻塞功能。

(3)在 Nexys Video 开发系统中实现 RISC-V 微处理器，要求 CPU 的运行速度大于 25MHz。

(二）扩展要求

（1）要求设计的微处理器还能运行 lb、lh、ld、lbu、lhu、lwu、sb、sh 或 sd 等字节、半字和双字数据传送指令。

（2）要求设计的 CPU 增加异常（exception）、自陷（trap）、中断（interrupt）等处理方案。

三、实验原理

（一） 总体设计

流水线是数字系统中一种提高系统稳定性和工作速度的方法，广泛应用于高档 CPU 的架构中。根据 RISC-V 处理器的特点，将指令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。如下图 1 所示，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。

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

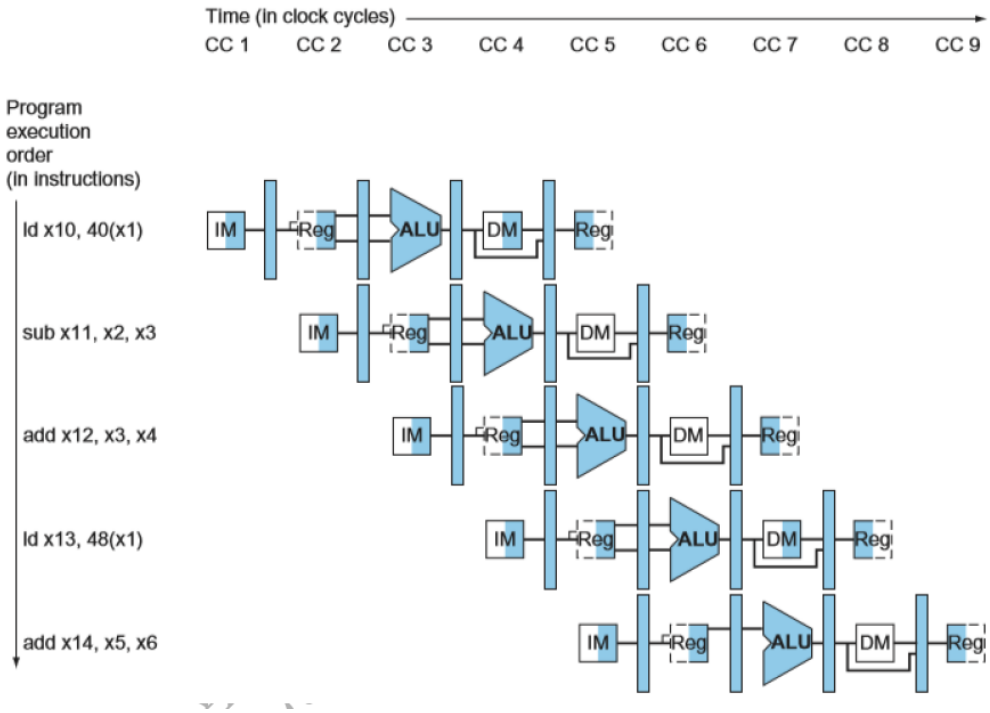


图 1 流水线流水作业示意图

由于在流水线中，数据和控制信息将在时钟周期的上升沿转移到下一级，所以规定流水线转移的变量命名遵守如下格式：名称_流水线级名称。如，在 ID 级指令译码电路（Decode）产生的寄存器写允许信号 RegWrite 在 ID 级、EX 级、MEM 级和 WB 级上的命名分别为 RegWrite_id、RegWrite_ex、RegWrite_mem 和 RegWrite_wb。在顶层文件中，类似的变量名称有近百个，这样的命名方式起到了很好的识别作用。

1. 流水线中的控制信号

（1）IF 级：取指令级。从 ROM 中读取指令，并在下一个时钟沿到来时把指令送到 ID 级的指令缓冲器中。该级控制信号有决定下一个指令指针的 PCSource 控制信号、阻塞 IF/ID 流水线同时暂停读取下一条指令的 IFWrite 信号、清空 IF/ID 寄存器的 IF_flush 信号。

（2）ID 级：指令译码器。对 IF 级来的指令进行译码，并产生相应的控制信号。整个 CPU 的控制信号基本都是在这级上产生。该级自身不需任何控制信号。

流水线冒险检测也在该级进行，即当流水线冒险条件成立时，冒险检测电路产生 Stall 信号清空 ID/EX 寄存器，同时冒险检测电路产生低电平 IFWrite 信号阻塞 IF/ID 流水线。即插入一个流水线气泡。

（3）EX 级：执行级。该级进行算术或逻辑操作。此外数据传送指令所用的 RAM 访问地址也是在本级上实现。控制信号有 ALUCode、ALUSrcA 和 ALUScrB，根据这些信号确定 ALU 操作、并选择两个 ALU 操作数 ALU_A、ALU_B。

另外，数据转发也在该级完成。数据转发控制电路产生 ForwardA 和 ForwardB 两组转发控制信号。

（4）MEM 级：存储器访问级。只有在执行数据传送指令时才对存储器进行读写，对其他指令只起到缓冲一个时钟周期的作用。该级只需存储器写操作允许信号 MemWrite。

（5）WB 级：回写级。此级把指令执行的结果回写到寄存器堆中。该级设置信号 MemtoReg 和寄存器写操作允许信号 RegWrite，其中 MemtoReg 决定写入寄存器的数据来

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

源：当 MemtoReg=0 时，回写数据来自 ALU 运算结果；而当 MemtoReg=1 时，回写数据来自存储器。

2. 数据相关与数据转发

如果上一条指令的结果还没有写入到寄存器中，而下一条指令的源操作数又恰恰是此寄存器的数据，那么，它所获得的将是原来的数据，而不是更新后的数据。这样的相关问题称为数据相关。如图 2 所示的五级流水结构，当前指令与前三条指令都构成数据相关问题。在设计中，采用数据转发和插入流水线气泡的方法解决此类相关问题。

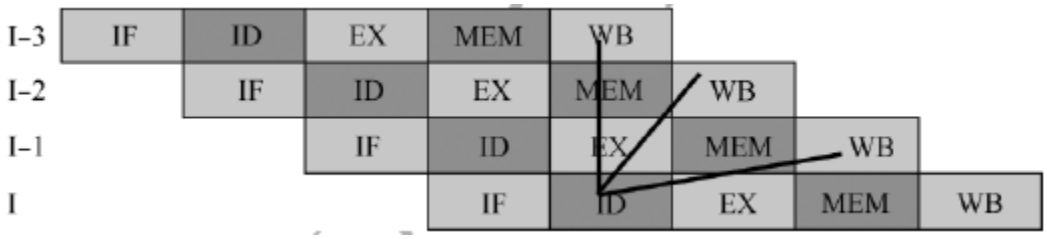


图 2 数据相关性问题示意图

（1） 一阶数据相关与转发（EX 冒险）

图 3 一阶前推网络示意图

如图 3 所示，如果源操作寄存器与第 I-1 条指令的目标操作寄存器相重，将导致一阶数据相关。从图 2 可以看出，第 I 条指令的 EX 级与第 I-1 条指令的 MEM 级处于同一时钟周期，且数据转发必须在第 I 条指令的 EX 级完成。因此，导致操作数 A 的一阶数据相关判断的条件为：

① MEM 级阶段必须是写操作（RegWrite_mem=1）；

② 目标寄存器不是 X0 寄存器（rdAddr_mem≠ 0）；

③ 两条指令读写同一个寄存器（rdAddr_mem=rs1Addr_ex）。

导致操作数 B 的一阶数据相关判断的条件为：

① MEM 级阶段必须是写操作（RegWrite_mem=1）；

② 目标寄存器不是 X0 寄存器（rdAddr_mem≠ 0）；

③ 两条指令读写同一个寄存器（rdAddr_mem=rs2Addr_ex）。

除了第 I-1 条指令为 lw 外，其它指令回写寄存器的数据均为 ALU 输出，因此当一阶数据相关时，除 lw 指令外，一阶数据相关的解决办法是将第 I-1 条指令的 MEM 级的 ALUResult_mem 转发至第 I 条 EX 级，如图 3 所示。

（2） 二阶数据相关与转发（MEM 冒险）

图 4 二阶前推网络示意图

如图 4 所示，如果第 I 条指令的源操作寄存器与第 I-2 指令的目标寄存器相重，将导致二阶数据相关。导致操作数 A 的二阶数据相关必须满足下列条件：

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

① WB 级阶段必须是写操作（RegWrite_wb=1）；
② 目标寄存器不是 X0 寄存器（rdAddr_wb≠ 0）；
③ 一阶数据相关条件不成立（rdAddr_mem≠rs1Addr_ex）；
④ 两条指令读写同一个寄存器（rdAddr_wb=rs1Addr_ex）。

导致操作数 B 的二阶数据相关必须满足下列条件：

① WB 级阶段必须是写操作（RegWrite_wb=1）；
② 目标寄存器不是 X0 寄存器（rdAddr_wb≠ 0）；
③ 一阶数据相关条件不成立（rdAddr_mem≠rs2Addr_ex）；
④ 两条指令读写同一个寄存器（rdAddr_wb=rs2Addr_ex）。

当发生二阶数据相关问题时，解决方法是将第 I-2 条指令的回写数据 RegWriteData 转发至第 I 条指令的 EX 级，如图 4 所示。

（3） 三阶数据相关

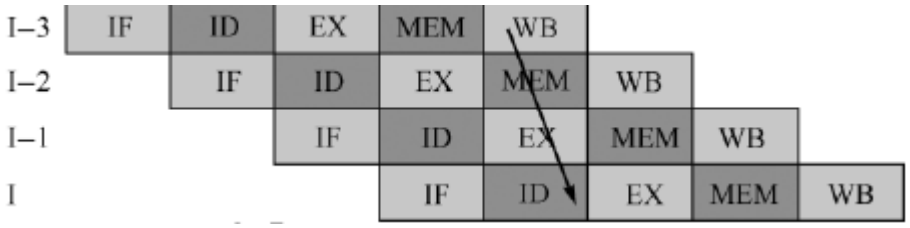


图 5 三阶前推网络示意图

图 5 所示为第 I 条指令与第 I-3 条指令的数据相关问题，即在同一个周期内同时读写同一个寄存器，将导致三阶数据相关。

导致操作数 A 的二阶数据相关必须满足下列条件：

① 寄存器必须是写操作（RegWrite_wb=1）；
② 目标寄存器不是 X0 寄存器（rdAddr_wb≠ 0）；
③ 读写同一个寄存器（rdAddr_wb=rs1Addr_id）。

导致操作数 B 的二阶数据相关必须满足下列条件：

① 寄存器必须是写操作（RegWrite_wb=1）；
② 目标寄存器不是 X0 寄存器（rdAddr_wb≠ 0）；
③ 读写同一个寄存器（rdAddr_wb=rs2Addr_id）。

该类数据相关问题可以通过改进设计寄存器堆的硬件电路来解决，要求寄存器堆具有 Read After Write 特性，即同一个周期内对同一个寄存器进行读、写操作时，要求读出的值为新写入的数据。

3. 数据相关与数据转发

当第 I 条指令读取一个寄存器，而第 I-1 条指令为 lw，且与 lw 写入为同一个寄存器时，定向转发是无法解决问题的。因此，当 lw 指令后跟一条需要读取它结果的指令时，必须采用相应的机制来阻塞流水线，即还需要增加一个冒险检测单元（Hazard Detector）。它工作在 ID 级，当检测到上述情况时，在 lw 指令和后一条指令之间插入气泡，使后一条指令延迟一个周期执行，这样可将一阶数据冒险问题变成二阶数据冒险问题，就可用转发解决。

冒险检测工作在 ID 级，前一条指令已处在 EX 级，冒险成立的条件为：

① 上一条指令必须是 lw 指令（MemRead_ex=1）；

装 订 线

② 两条指令读写同一个寄存器（rdAddr_ex=rs1Addr_id 或 rdAddr_ex=rs2Addr_id）。

当上述条件满足时，指令将被阻塞一个周期，Hazard Detector 电路输出的 Stall 信号清空 ID/EX 寄存器，另外一个输出低电平有效的 IFWrite 信号阻塞流水线 ID 级、IF 级，即插入一个流水线气泡。

（二） 流水线 RISC-V 微处理器的设计

根据流水线不同阶段，将系统划分为 IF、ID、EX 和 MEM 四大模块，WB 部分功能电路非常简单，因而采取在顶层文件中设计的方法。另外，系统还包括 IF/ID、ID/EX、EX/MEM、MEM/WB 四个流水线寄存器。

1. 指令译码模块（ID）的设计

指令译码模块的主要作用是从机器码中解析出指令，并根据解析结果输出各种控制信号。ID 模块主要有指令译码（Decode）、寄存器堆（Registers）、冒险检测、分支检测和加法器等组成。ID 模块的接口信息如下表所示：

表 1 ID 模块的输入/输出引脚说明

引脚名称	方向	说　　明
clk	Input	系统时钟
Instruction_id[31:0]		指令机器码
PC_id[31:0]		指令指针
RegWrite_wb		寄存器写允许信号，高电平有效
rdAddr_wb[4:0]		寄存器的写地址
RegWriteData_wb[31:0]		写入寄存器的数据
MemRead_ex		冒险检测的输入
rdAddr_ex[4:0]		
MemtoReg_id	Output	决定回写的数据来源（0：ALU　1：存储器）
RegWrite_id		寄存器写允许信号，高电平有效
MemWrite_id		存储器写允许信号，高电平有效
MemRead_id		存储器读允许信号，高电平有效
ALUCode_id[4:0]		决定 ALU 采用何种运算
ALUSrcA_id		决定 ALU 的 A 操作数的来源（0：rs1；1：pc）
ALUSrcB_id[1:0]		决定 ALU 的 B 操作数的来源（2'b00：rs2；2'b01：imm；2'b10：常数 4）
Stall		ID/EX 寄存器清空信号，高电平表示插入一个流水线气泡
Branch		条件分支指令的判断结果，高电平有效
Jump		无条件分支指令的判断结果，高电平有效
IFWrite		阻塞流水线的信号，低电平有效
BranchAddr[31:0]		条件分支地址
Imm_id[31:0]		立即数
rdAddr_id[4:0]		回写寄存器地址
rs1Addr_id[4:0]		两个数据寄存器地址
rs2Addr_id[4:0]		
rs1Data_id[31:0]		寄存器两个端口输出数据
rs2Data_id[31:0]		

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

(1) 寄存器堆（Registers）的设计

寄存器堆由 32 个 32 位寄存器组成，这些寄存器通过寄存器号进行读写存取。寄存器堆的原理框图如图 6 所示。因为读取寄存器不会更改其内容，故只需提供寄存号即可读出该寄存器内容。读取端口采用数据选择器即可实现读取功能。应注意“0”号寄存器为常数 0.

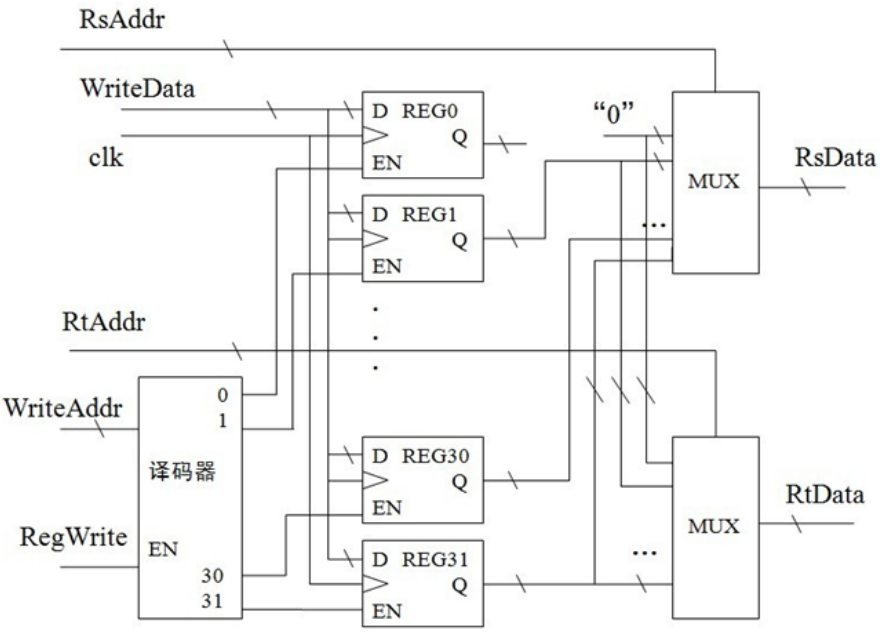


图 6 寄存器堆的原理框图

对于往寄存器里写数据，需要目标寄存器号（WriteRegister）、待写入数据（WriteData）、写允许信号（RegWrite）三个变量。图 6 中 5 位二进制译码器完成地址译码，其输出控制目标寄存器的写使能信号 EN，决定将数据 WriteData 写入哪个寄存器。

描述寄存器堆核心语句；

reg [31:0] regs[31:0];//定义 32*32 存储器变量

assign ReadData1=(ReadRegister1==5’b0)?32’b0:regs[ReadRegister1];//从端口 1 数据读出

assign ReadData2=(ReadRegister2==5’b0)?32’b0:regs[ReadRegister2];//从端口 2 数据读出

always @(posedge clk) if(RegWrite) regs[WriteRegsiter]<=WriteData;//数据写入

在流水线 CPU 设计中，寄存器堆设计还应解决三阶数据相关的数据转发问题。当满足三阶数据相关条件时，寄存器具有 Read After Write 的特性。为实现该功能，只需要在图 6 设计寄存器堆的基础上添加少量电路就可实现 Read After Write 的特性，如图 7 所示。图中的 RBW_Registers 模块就是实现图 6 中的 Read Before Write 寄存器堆。图中转发检测电路的输出表达式为

rs1Sel=RegWrite&&(WriteAddr!=0)&&(WriteAddr==rs1Addr)

rs2Sel=RegWrite&&(WriteAddr!=0)&&(WriteAddr==rs2Addr)

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

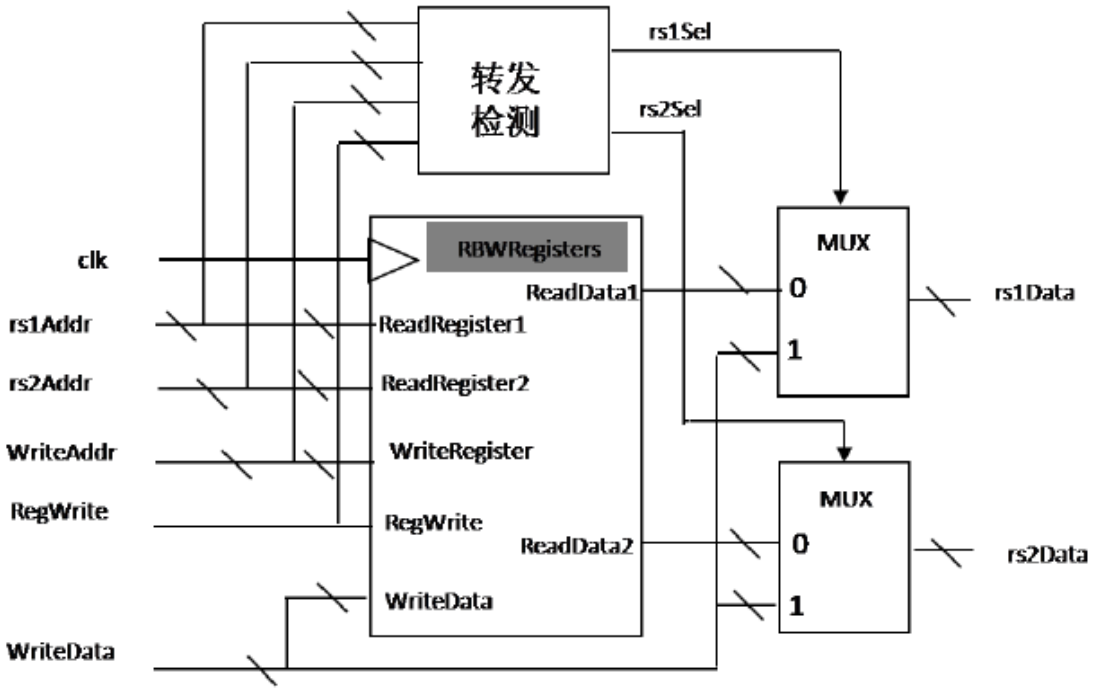


图 7 具有 Read After Write 特性寄存器堆的原理框图

(2) 指令译码（包含立即数产生电路）子模块的设计

该子模块主要作用是根据指令确定各个控制信号的值，同时产生立即数 Imm 和偏移量 offest。该模块是一个组合电路。

RISC-V 将指令分为 R、I、S、SB、U、UJ 等六类。从电路设计角度看，根据操作数的来源和立即数构成方式不同，再次细分指令。具体做法：

- R_type 类：操作码（opcode，简称 op）为 7'h33，R 类的所有指令，两个操作数分别为 rs1 和 rs2；
- I_type 类：操作码 7'h13，I 类的算术逻辑运算指令和移位指令，两个操作数分别为 rs1 和立即数 imm；
- LW 指令：操作码 7'h03，I 类的数据传送指令 lw，两个操作数分别为 rs1 和立即数 imm；
- JALR 指令：操作码 7'h67，I 类的无条件分支指令 jalr，两个操作数分别为 PC 和常数 4；
- SW 指令：操作码 7'h23，S 类的数据传送指令 sw，两个操作数分别为 rs1 和立即数 imm；
- SB_ type 类：操作码 7'h63，SB 类的所有指令，两个操作数分别为 PC 和立即数 imm；
- LUI 指令：操作码 7'h37，U 类的数据传送指令 lui，只有一个操作数（立即数 imm)；
- AUIPC 指令：操作码 7'h17，U 类的数据传送指令 auipc，两个操作数分别为 PC 和立即数 imm；
- JAL 指令：操作码 7'h6F，UJ 类的无条件分支指令 jal，两个操作数分别为 PC 和常数 4。

因此，设置 R_type、I_type、SB_type、LW、JALR、SW、LUI、AUIPC、和 JAL 等变量来表示，各变量的值由下式决定。

R_type=(op==R_type_op)
I_type=(op==I_type_op)
SB_type=(op==SB_type_op)
LW=(op==LW_op)
JALR=(op==JALR_op)
SW=(op==SW_op)

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

LUI=(op==LUI_op)

AUIPC=(op==AUIPC_op)

JAL=(op==JAL_op)

① 只有 LW 指令读取存储器且回写数据取自存储器，所以有

MemtoReg_id=LW

MemRead_id=LW

② 只有 SW 指令会对存储器写数据，所以有

MemWrite_id=SW

③ 需要回写的指令类型有 R_type、I_type、LW、JALR、LUI、AUIPC、和 JAL。所以有 RegWrite_id=R_type||I_type||LW||JALR||LUI||AUIPC||JAL

④ 只有 JALR 和 JAL 两条无条件分支指令，所以有

Jump=JALR||JAL

⑤ 操作数 A 和 B 的选择信号的确定

分析各类指令，可得到以下操作数选择的功能表。

类型	ALUSrcA_id	ALUSrcB_id[1:0]	说明
R_type	0	2'b00	rd=rs1 op rs2
I_type	0	2'b 01	rd=rs1 op imm
LW	0	2'b 01	rs1 + imm
SW	0	2'b 01	rs1 + imm
JALR	1	2'b 10	rd=pc + 4
JAL	1	2'b 10	rd=pc + 4
LUI	1'bx	2'b 01	rd= imm
AUIPC	1	2'b 01	rd=pc + imm

从表中可获得 ALUSrcA_id 和 ALUSrcB_id 表达式。

ALUSrcA_id=JALR||JAL||AUIPC

ALUSrcB_id[1]=JAL||JALR

ALUSrcB_id[0]=~(R_type||JAL||JALR)

⑥ ALUCode 的确定

除了条件分支指令，其它指令都需要 ALU 执行运算，共有 11 种不同运算，ALUCode 信号需用 4 位二进制表示。最主要为加法运算，设为默认算法，ALUCode 的功能表如下表所示。

R_type	I_type	LUI	funct3	funct7[6] (funct6[5])	ALUCode	备注
1	0	0	3'o0	0	4'd 0	加
1	0	0	3'o0	1	4'd 1	减
1	0	0	3'o1	0	4'd 6	左移 A << B
1	0	0	3'o2	0	4'd9	A<B?1:0
1	0	0	3'o3	0	4'd10	A<B?1:0（无符号数）
1	0	0	3'o4	0	4'd 4	异或
1	0	0	3'o5	0	4'd 7	右移 A >> B
1	0	0	3'o5	1	4'd 8	算术右移 A >>>B
1	0	0	3'o6	0	4'd 5	或
1	0	0	3'o7	0	4'd 3	与
0	1	0	3'o0	x	4'd 0	加
0	1	0	3'o1	x	4'd 6	左移
0	1	0	3'o2	x	4'd9	A<B?1:0
0	1	0	3'o3	x	4'd10	A<B?1:0（无符号数）
0	1	0	3'o4	x	4'd 4	异或
0	1	0	3'o5	0	4'd 7	右移 A >> B
0	1	0	3'o5	1	4'd 8	算术右移 A >>>B
0	1	0	3'o6	x	4'd 5	或
0	1	0	3'o7	x	4'd 3	与
0	0	1	x	x	4'd 2	送数:ALUResult=B
其它					4'd 0	加

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

⑦ 立即数产生电路（ImmGen）设计

I_type、SB_type、LW、JALR、SW、LUI、AUIPC、和 JAL 这几类指令均用到了立即数。由于 I_type 的算术逻辑运算与移位运算指令的立即数构成方法不同，这里再设定一个变量 shift 来区分两者。Shift=1 表示移位运算，否则为算术逻辑运算。Shift 值由下式计算。

Shift=(funct3==1)||(funct3==5)

立即数构成和扩展方法如表所示：

类别	Shift	Imm	offset
I_type	1	{26'd0,inst[25:20]}	-
I_type	0	{{20{inst[31]}},inst[31:20]}	-
LW	x		-
JALR	x	-	{{20{inst[31]}},inst[31:20]}
SW	x	{{20{inst[31]}},inst[31:25],inst[11:7]}	-
JAL	x	-	{{11{inst[31]}}, inst[31], inst[19:12] , inst[20], inst[30:21], 1'b0}
LUI	x	{inst[31:12], 12'd0}	-
AUIPC	x		-
SB_type	x	-	{{19{inst[31]}}, inst[31], inst[7],inst[30:25] , inst[11:8], 1'b0}

（3） 分支检测（Branch Test）电路的设计

分支检测电路主要用来判断分支条件是否成立，在 Verilog HDL 可以用比较运算符号“>”、“=”和“<”描述，但要注意符号数和无符号数的处理方法不同。在这里，我们用加法器来实现。

① 用 32 位加法器完成 rs1Data+(~rs2Data)+1(即 rs1Data-rs2Data)，设结果为 sum[31:0]。

② 确定比较运算的结果，对于比较运算来说，如果最高位不同，即 rs1Data[31]≠rs2Data[31]，可根据 rs1Data[31]、rs2Data[31]决定比较结果，但是应注意符号数、无符号数的最高位代表意义不同。若两数最高位相同，则两数之差不会溢出，所以比较运算结果可由两个操作数之差的符号位 sum[31]决定。

符号数：isLT=rs1Data[31]&&(~rs2Data[31])||(rs1Data[31]~^rs2Data[31])&&sum[31]

无符号数：

isLTU=(~rs1Data[31])&&rs2Data[31]||(rs1Data[31]~^rs2Data[31])&&sum[31]

最后用数据选择器完成下式即可。

$$Branch = \begin{cases} \sim(|sum[31:0]); & SB\_type \ \&\& \ (funct3 == beq\_funct3) \\ |sum[31:0]; & SB\_type \ \&\& \ (funct3 = bne\_funct3) \\ isLT; & SB\_type \ \&\& (funct3 = blt\_funct3) \\ \sim isLT; & SB\_type \ \&\& \ (funct3 = bge\_funct3) \\ isLTU; & SB\_type \ \&\& \ (funct3 = bltu\_funct3) \\ \sim isLTU; & SB\_type \ \&\& \ (funct3 = bgeu\_funct3) \\ 0 & others \end{cases}$$

（4） 冒险检测功能电路（Hazard Detector）的设计

由前面分析可知，冒险成立的条件为：

① 上一条指令必须是 lw 指令（MemRead_ex=1）；

装 订 线

② 两条指令读写同一个寄存器（rdAddr_ex=rs1Addr_id 或 rdAddr_ex=rs2Addr_id）。

当冒险成立应清空 ID/EX 寄存器并且阻塞流水线 ID 级、IF 级流水线，所以有 Stall=((rdAddr_ex==rs1Addr_id)||(rdAddr_ex==rs2Addr_id))&&MemRead_ex IFWrite=~Stall

2. 执行模块（EX）的设计

执行模块主要由 ALU 子模块、数据前推电路（Forwarding）及若干数据选择器组成。执行模块的接口信息如下表所示。

引脚名称	方向	说明
ALUCode_ex[3:0]	Input	决定 ALU 采用何种运算
ALUSrcA_ex		决定 ALU 的 A 操作数的来源（rs1、PC）
ALUSrcB_ex[1:0]		决定 ALU 的 B 操作数的来源(rs2、imm 和常数 4)
Imm_ex[31:0]		立即数
rs1Addr_ex[4:0]		rs1 寄存器地址
rs2Addr_ex[4:0]		rs2 寄存器地址
rs1Data_ex[31:0]		rs1 寄存器数据
rs2Data_ex[31:0]		rs2 寄存器数据
PC_ex[31:0]		指令指针
RegWriteData_wb[31:0]		写入寄存器的数据
ALUResult_mem[31:0]		ALU 输出数据
rdAddr_mem[4:0]		寄存器的写地址
rdAddr_wb[4:0]		
RegWrite_mem		寄存器写允许信号
RegWrite_wb		
ALUResult_ex[31:0]	Output	ALU 运算结果
MemWriteData_ex[31:0]		存储器的回写数据
ALU_A [31:0]		ALU 操作数，测试时使用
ALU_B [31:0]		

（1） ALU 子模块的设计

算术逻辑单元（ALU）提供 CPU 的基本运算能力，如加、减、与、或、比较、位移等。具体而言，ALU 输入为两个操作数 A、B 和控制信号 ALUCode，由控制信号 ALUCode 决定采用何种运算，运算结果为 ALUResult。下表为 ALU 功能表。

ALUCode	ALUResult
4'b0000	A + B
4'b0001	A-B
4'b0010	B
4'b0011	A&B
4'b0100	A ^ B
4'b0101	A | B
4'b0110	A << B
4'b0111	A >> B
4'b1000	A>>>B
4'b1001	A<B? 1:0，其中 A、B 为有符号数
4'b1010	A<B? 1:0，其中 A、B 为无符号数

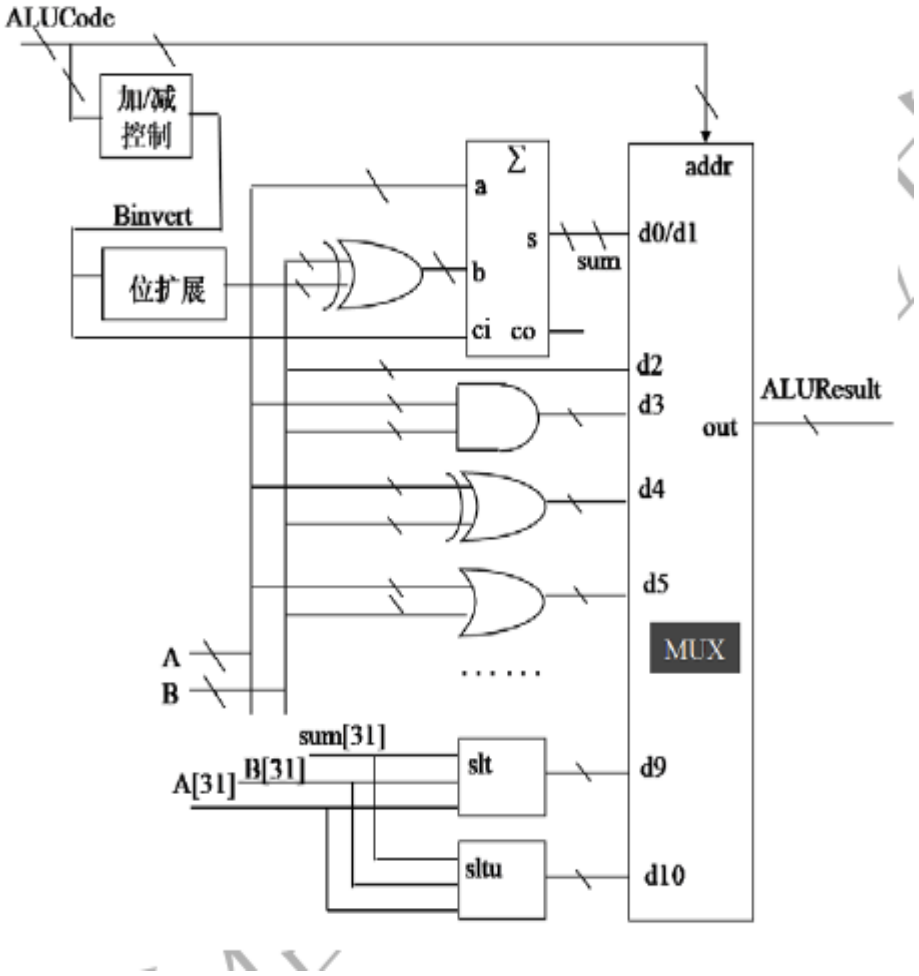
实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

如上表所示，ALU 需执行多种运算，为了提高运算速度，本设计可同时进行多种运算，再根据 ALUCode 信号选出所需结果。

① 加、减电路的设计考虑

减法、比较（slt、sltu）均可用加法器和必要辅助电路来实现。Binvert 信号控制加减运算；若 Binvert 信号为低电平，则进行加法运算；sum=A+B；若 Binvert 信号为高电平，则电路为减法运算 sum=A-B。除加法外，减法、比较和分支指令都应使电路工作再减法状态，所以：

Binvert=~(ALUCode==0)



② 比较电路的设计考虑

比较电路的设计方法已在分支检测电路中说明。

③ 算术右移运算电路的设计考虑

算术右移对于有符号数而言，移出的高位补符号位而不是 0。每右移一位相当于除以 2。

Verilog HDL 的算术右移的运算符是“>>>”。要实现算术右移应注意，被移位的对象必须定义为 reg 类型，但是在 sra 指令，被移位的对象操作数 A 为输入信号，不能定义为 reg 类型。因此，必须引入 reg 类型中间变量 A_reg，相应的 Verilog HDL 语句为 reg signed [31:0] A_reg; always @(*)begin A_reg=A; end

(2) 数据前推电路的设计

操作数 A 和 B 分别由数据选择器决定，数据选择器地址信号如下表所示：

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

地　址	操作数来源	说　明
ForwardA= 2'b00	rs1Data_ex	操作数 A 来自寄存器堆
ForwardA=2'b01	RegWriteData_wb	操作数 A 来自二阶数据相关的转发数据
ForwardA= 2'b10	ALUResult_mem	操作数 A 来自一阶数据相关的转发数据
ForwardB= 2'b00	rs2Data_ex	操作数 B 来自寄存器堆
ForwardB= 2'b01	RegWriteData_wb	操作数 B 来自二阶数据相关的转发数据
ForwardB=2'b10	ALUResult_mem	操作数 B 来自一阶数据相关的转发数据

由前面介绍的一、二阶数据相关判断条件，不难得到

$$\begin{cases} ForwardA[0] = RegWrite\_wb \,\&\&\, (rdAddr\_wb != 0) \,\&\&\, (rdAddr\_mem != rs1Addr\_ex) \,\&\&\, (rdAddr\_wb == rs1Addr\_ex) \\ ForwardA[1] = RegWrite\_mem \,\&\&\, (rdAddr\_mem != 0) \,\&\&\, (rdAddr\_mem == rs1Addr\_ex) \end{cases}$$

$$\begin{cases} ForwardB[0] = RegWrite\_wb \,\&\&\, (rdAddr\_wb != 0) \,\&\&\, (rd\_mem != rs2Addr\_ex) \,\&\&\, (rdAddr\_wb == rs2Addr\_ex) \\ ForwardB[1] = RegWrite\_mem \,\&\&\, (rdAddr\_mem != 0) \,\&\&\, (rdAddr\_mem == rs2Addr\_ex) \end{cases}$$

装

订

线

3. 数据存储器模块的设计

数据存储器可用 Xilinx 的 IP 内核实现，设计为容量为 64*32bit 的单端口 RAM，输出采用组合输出。地址与 ALUResult[7:2]链接。

4. 取指令级模块的设计

IF 模块由指令寄存器 PC、指令存储器子模块（Instruction ROM）、指令选择器（MUX）和一个 32 位加法器组成，接口信息如下。

引脚名称	方向	说明
clk	Input	系统时钟
reset		系统复位信号，高电平有效
Branch		条件分支指令的条件判断结果
Jump		无条件分支指令的条件判断结果
IFWrite		流水线阻塞信号
JumpAddr[31:0]		分支指令跳转地址
Instruction [31:0]	Output	指令机器码
IF_flush		流水线清空信号
PC [31:0]		PC 值

5. 流水线寄存器的设计

流水线寄存器负责将流水线的各部分分开，共有 IF/ID、ID/EX、EX/MEM、MEM/WB 四组，对四组流水线寄存器要求不完全相同，因此设计也有不同考虑。

EX/MEM、MEM/WB 两组流水线寄存器只是普通的 D 型寄存器。

当流水线发生数据冒险时，需要清空 ID/EX 流水线寄存器而插入一个气泡，因此 ID/EX 流水线寄存器是一个带同步清零功能的 D 型寄存器。

当流水线发生数据冒险时，需要阻塞 IF/ID 流水线寄存器；若跳转指令或分支成立，则还需要清空 ID/EX 流水线寄存器。因此，IF/ID 流水线寄存器除同步清零功能外，还需要具有保持功能（即具有使能 EN 信号输入）。

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

四、Verilog HDL 代码及分析

代码分析已包含在注释中。

1. 顶层代码

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: zju
// Engineer: qmj
//////////////////////////////////////////////////////////////////////////////////
module Risc5CPU(clk, reset, JumpFlag, Instruction_id, ALU_A,
                                ALU_B, ALUResult_ex, PC, MemDout_mem,Stall);
    input clk;
    input reset;
    output [1:0] JumpFlag;
    output [31:0] Instruction_id;
    output [31:0] ALU_A;
    output [31:0] ALU_B;
    output [31:0] ALUResult_ex;
    output [31:0] PC;
    output [31:0] MemDout_mem;
    output Stall;
////////////////////////////////中间连线的变量的定义//////////////////////////////////
    wire Branch,Jump,IFWrite;
    wire [31:0] JumpAddr,Instruction_if,PC_if;
    wire IF_flush;

    wire [31:0] PC_id;

    wire RegWrite_id,MemtoReg_id,MemWrite_id,MemRead_id;
    wire [3:0] ALUCode_id;
    wire ALUSrcA_id;
    wire [1:0] ALUSrcB_id;
    wire [31:0] Imm_id;
    wire [4:0] rdAddr_id,rs1Addr_id,rs2Addr_id;
    wire [31:0] rs1Data_id,rs2Data_id;

    wire MemtoReg_ex,RegWrite_ex,MemWrite_ex,MemRead_ex;
    wire [3:0] ALUCode_ex;
    wire ALUSrcA_ex;
    wire [1:0] ALUSrcB_ex;
    wire [31:0] PC_ex,Imm_ex,rs1Data_ex,rs2Data_ex,MemWriteData_ex;
    wire [4:0] rdAddr_ex,rs1Addr_ex,rs2Addr_ex;

    wire MemtoReg_mem,RegWrite_mem,MemWrite_mem;
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
    wire [31:0] ALUResult_mem,MemWriteData_mem;
    wire [4:0] rdAddr_mem;

    wire [31:0] MemDout_wb,ALUResult_wb,RegWriteData_wb;
    wire MemtoReg_wb,RegWrite_wb;
    wire [4:0] rdAddr_wb;
////////////////////对于输出 JumpFlag 及 PC 的值进行定义//////////////////////////////////////
    assign JumpFlag={Jump,Branch};
    assign PC=PC_if;

///////////////IF//////////////////////////////
///////////////输入值 Branch,Jump,IFWrite 带有 reset 的初始化值////////////////////
    IF inst1(
        .clk(clk),
        .reset(reset),
        .Branch(Branch|reset),
        .Jump(Jump|reset),
        .IFWrite(IFWrite&(~reset)),
        .JumpAddr(JumpAddr),
        .Instruction_if(Instruction_if),
        .IF_flush(IF_flush),
        .PC(PC_if));

////////IF/ID//////////////////////////////
////////已置入 reset，可一键清 0////////////////////
    dffre #(.n(32)) inst11(
        .d(PC_if),
        .en(IFWrite),
        .r(IF_flush|reset),
        .clk(clk),
        .q(PC_id));
    dffre #(.n(32)) inst12(
        .d(Instruction_if),
        .en(IFWrite),
        .r(IF_flush|reset),
        .clk(clk),
        .q(Instruction_id));

////////ID//////////////////////////////

    ID inst2(
        .clk(clk),
        .Instruction_id(Instruction_id),
```

装 订 线

```
        PC_id(PC_id),
        .RegWrite_wb(RegWrite_wb),
        .rdAddr_wb(rdAddr_wb),
        .RegWriteData_wb(RegWriteData_wb),
        .MemRead_ex(MemRead_ex),
        .rdAddr_ex(rdAddr_ex),
        .MemtoReg_id(MemtoReg_id),
        .RegWrite_id(RegWrite_id),
        .MemWrite_id(MemWrite_id),
        .MemRead_id(MemRead_id),
        .ALUCode_id(ALUCode_id),
        .ALUSrcA_id(ALUSrcA_id),
        .ALUSrcB_id(ALUSrcB_id),
        .Stall(Stall),
        .Branch(Branch),
        .Jump(Jump),
        .IFWrite(IFWrite),
        .JumpAddr(JumpAddr),
        .Imm_id(Imm_id),
        .rs1Data_id(rs1Data_id),
        .rs2Data_id(rs2Data_id),
        .rs1Addr_id(rs1Addr_id),
        .rs2Addr_id(rs2Addr_id),
        .rdAddr_id(rdAddr_id));

////////ID/EX//////////////////////////////
////////已置入 reset，可一键清 0//////////////////////
    dffre   #(.n(1)) inst21(
        .d(MemtoReg_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(MemtoReg_ex));
    dffre   #(.n(1)) inst22(
        .d(RegWrite_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(RegWrite_ex));
    dffre   #(.n(1)) inst23(
        .d(MemWrite_id),
        .en(1'b1),
```

装 订 线

```
        .r(Stall|reset),
        .clk(clk),
        .q(MemWrite_ex));
    dffre   #(.n(1)) inst24(
        .d(MemRead_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(MemRead_ex));
    dffre   #(.n(4)) inst25(
        .d(ALUCode_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(ALUCode_ex));
    dffre   #(.n(1)) inst26(
        .d(ALUSrcA_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        . q(ALUSrcA_ex));
    dffre   #(.n(2)) inst27(
        .d(ALUSrcB_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(ALUSrcB_ex));
    dffre   #(.n(32)) inst28(
        .d(PC_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(PC_ex));
    dffre   #(.n(32)) inst29(
        .d(Imm_id),
        .en(1'b1),
        .r(Stall|reset),
        .clk(clk),
        .q(Imm_ex));
    dffre   #(.n(5)) inst291(
        .d(rdAddr_id),
        .en(1'b1),
        .r(Stall|reset),
```

装 订 线

```
            .clk(clk),
            .q(rdAddr_ex));
        dffre    #(.n(5)) inst292(
            .d(rs1Addr_id),
            .en(1'b1),
            .r(Stall|reset),
            .clk(clk),
            .q(rs1Addr_ex));
        dffre    #(.n(5)) inst293(
            .d(rs2Addr_id),
            .en(1'b1),
            .r(Stall|reset),
            .clk(clk),
            .q(rs2Addr_ex));
        dffre    #(.n(32)) inst294(
            .d(rs1Data_id),
            .en(1'b1),
            .r(Stall|reset),
            .clk(clk),
            .q(rs1Data_ex));
        dffre    #(.n(32)) inst295(
            .d(rs2Data_id),
            .en(1'b1),
            .r(Stall|reset),
            .clk(clk),
            .q(rs2Data_ex));

    ////////EX//////////////////////////////////

        EX inst3(
            .ALUCode_ex(ALUCode_ex),
            .ALUSrcA_ex(ALUSrcA_ex),
            .ALUSrcB_ex(ALUSrcB_ex),
            .Imm_ex(Imm_ex),
            .rs1Addr_ex(rs1Addr_ex),
            .rs2Addr_ex(rs2Addr_ex),
            .rs1Data_ex(rs1Data_ex),
            .rs2Data_ex(rs2Data_ex),
            .PC_ex(PC_ex),
            .RegWriteData_wb(RegWriteData_wb),
            .ALUResult_mem(ALUResult_mem),
            .rdAddr_mem(rdAddr_mem),
            .rdAddr_wb(rdAddr_wb),
```

装 订 线

```
        .RegWrite_mem(RegWrite_mem),
        .RegWrite_wb(RegWrite_wb),
        .ALUResult_ex(ALUResult_ex),
        .MemWriteData_ex(MemWriteData_ex),
        .ALU_A(ALU_A),
        .ALU_B(ALU_B));

////////EX/MEM////////////////////////////////

    dffre   #(.n(1)) inst31(
        .d(MemtoReg_ex),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(MemtoReg_mem));
    dffre   #(.n(1)) inst32(
        .d(RegWrite_ex),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(RegWrite_mem));
    dffre   #(.n(1)) inst33(
        .d(MemWrite_ex),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(MemWrite_mem));
    dffre   #(.n(32)) inst34(
        .d(ALUResult_ex),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(ALUResult_mem));
    dffre   #(.n(32)) inst35(
        .d(MemWriteData_ex),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(MemWriteData_mem));
    dffre   #(.n(5)) inst36(
        .d(rdAddr_ex),
        .en(1'b1),
        .r(reset),
```

装 订 线

```
        .clk(clk),
        .q(rdAddr_mem));
////////DataRAM//////////////////////////////

    DataRAM inst6(
        .a(ALUResult_mem[7:2]),
        .d(MemWriteData_mem),
        .clk(clk),
        .we(MemWrite_mem),
        .spo(MemDout_mem));

////////MEM/WB//////////////////////////////

    dffre #(.n(1)) inst41(
        .d(MemtoReg_mem),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(MemtoReg_wb));
    dffre  #(.n(1)) inst42(
        .d(RegWrite_mem),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(RegWrite_wb));
   dffre #(.n(32)) inst43(
        .d(MemDout_mem),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(MemDout_wb));
    dffre  #(.n(32)) inst44(
        .d(ALUResult_mem),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(ALUResult_wb));
    dffre #(.n(5)) inst45(
        .d(rdAddr_mem),
        .en(1'b1),
        .r(reset),
        .clk(clk),
        .q(rdAddr_wb));
```

装 订 线

```
    mux32 inst7 (
        .out(RegWriteData_wb),
        .in0(ALUResult_wb),
        .in1(MemDout_wb),
        .addr(MemtoReg_wb));

endmodule
```

2. IF 模块代码

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:    zju
// Engineer: qmj
//////////////////////////////////////////////////////////////////////////////////
module IF(clk, reset, Branch,Jump, IFWrite, JumpAddr,Instruction_if,IF_flush,PC);
    input clk;
    input reset;
    input Branch;
    input Jump;
    input IFWrite;
    input [31:0] JumpAddr;
    output [31:0] Instruction_if;
    output IF_flush;
    output [31:0] PC;

    wire PCSourse;
    assign PCSourse=Jump|Branch;

    wire [31:0] NextPC_if;
    wire [31:0] PC1;
//////////////////////加法器，实现 PC+4////////////////////////////////
    adder_32bits inst1(
        .a(PC),
        .b(32'b100),
        .ci(1'b0),
        .s(NextPC_if),
        .co());
//////////////////////////定义 IF_flush/////////////////////////////////////////
    assign IF_flush=PCSourse;
//////////////////////PCSource 使能，选择 PC 来源////////////////////
    mux32 inst4 (
        .out(PC1),
        .in0(NextPC_if),
        .in1(JumpAddr),
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
        .addr(PCSourse));
//////////////////IFWrite 使能，PC 寄存器////////////////////////
    dffre #(.n(32)) inst2(
        .d(PC1),
        .en(IFWrite),
        .r(reset),
        .clk(clk),
        .q(PC));
////////////////////取指//////////////////////////////
    InstructionROM inst3(
        .addr(PC[7:2]),
        .dout(Instruction_if));

endmodule
```

2.1. InstructionROM 模块代码

```
/*-----------------------------------------------------------------

        lui X30,0x3000
        jalr X31 later(X0)
earlier:sw   X28, 0C(X0)
        lw   X29, 04(X6)
        sll   X5, X29, 2          //数据冒险
        lw    X28, 04(X6)
        sltu X28,X6,X7
done:   jal X31,done
later:  bne X0, X0, end          // 分支条件不成立
        addi X5, X30, 42
        add   X6, X0, X31
        sub X7, X5, X6                //操作 A 一阶数据相关，操作 B 二阶数据相关
        or    X28, X7, X5            //操作 A 一阶数据相关，操作 B 三阶数据相关
        beq X0, X0, earlier      // 分支条件成立
end:    nop

-----------------------------------------------------------------------*/

module InstructionROM(addr,dout);
    input [5 : 0] addr;
    output [31 : 0] dout;
    //
    reg [31 : 0] dout;
    always @(*)
        case (addr)
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
                6'd0:    dout=32'h0000_3f37 ;//                lui X30,0x3000
                6'd1:    dout=32'h0200_0fE7 ;//                jalr X31 later(X0)
                6'd2:    dout=32'h01c0_2623 ;// earlier: sw    X28, 0C(X0)
                6'd3:    dout=32'h0043_2e83 ;//                lw    X29, 4(X6)
                6'd4:    dout=32'h002e_9293 ;//                sll    X5, X29, 2
                6'd5:    dout=32'h0043_2e03 ;//                lw      X28, 4(X6)
                6'd6:    dout=32'h0073_3e33 ;//                sltu X28,X6,X7
                6'd7:    dout=32'h0000_0f6f ;//    done:      jal X31,done
                6'd8:    dout=32'h0000_1c63 ;//    later:      bne X0, X0, end          // 分支条件
不成立
                6'd9:    dout=32'h042f_0293;//                 addi X5, X30, 42
                6'd10:   dout=32'h01f0_0333 ;//                add    X6, X0, X31
                6'd11:   dout=32'h4062_83b3 ;//                 sub X7, X5, X6
                6'd12:   dout=32'h0053_ee33 ;//                or      X28, X7, X5
                6'd13:   dout=32'hfc00_0ae3 ;//               beq X0, X0, earlier
                6'd14: dout=32'h00000000 ;// end:        nop
                default:dout=32'h00000000 ;//nop
            endcase
endmodule
```

3. ID 模块代码

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//////////////////////////////////////////////////////////////////////////////////
module ID(clk,Instruction_id, PC_id, RegWrite_wb, rdAddr_wb, RegWriteData_wb,
MemRead_ex,
                rdAddr_ex, MemtoReg_id, RegWrite_id, MemWrite_id, MemRead_id, ALUCode_id,
                    ALUSrcA_id, ALUSrcB_id,    Stall, Branch, Jump, IFWrite,    JumpAddr,
Imm_id,
                    rs1Data_id, rs2Data_id,rs1Addr_id,rs2Addr_id,rdAddr_id);
    input clk;
    input [31:0] Instruction_id;
    input [31:0] PC_id;
    input RegWrite_wb;
    input [4:0] rdAddr_wb;
    input [31:0] RegWriteData_wb;
    input MemRead_ex;
    input [4:0] rdAddr_ex;



    output MemtoReg_id;
    output RegWrite_id;
```

```
    output MemWrite_id;
    output MemRead_id;
    output [3:0] ALUCode_id;
    output ALUSrcA_id;
    output [1:0]ALUSrcB_id;
    output Stall;
    output Branch;
    output Jump;
    output IFWrite;
    output [31:0] JumpAddr;
    output [31:0] Imm_id;
    output [31:0] rs1Data_id;
    output [31:0] rs2Data_id;
    output [4:0] rs1Addr_id,rs2Addr_id,rdAddr_id;

    wire [6:0]op;
    assign op=Instruction_id[6:0];
//////////////////////由 Instruction_id 直接得出 rs1Addr_id,rs2Addr_id,rdAddr_id//////////////////
    assign
rs1Addr_id=((op==7'h33)||(op==7'h13)||(op==7'h03)||(op==7'h23)||(op==7'h67)||(op==7'h63))?Ins
truction_id[19:15]:5'b0;
    assign rs2Addr_id=((op==7'h33)||(op==7'h23)||(op==7'h63))?Instruction_id[24:20]:5'b0;
    assign
rdAddr_id=((op==7'h33)||(op==7'h13)||(op==7'h03)||(op==7'h37)||(op==7'h17)||(op==7'h6f)||(op=
=7'h67))?Instruction_id[11:7]:5'b0;

////////////////////////////////////////////////////////////////////////////////
// Decode and Imm Gen
////////////////////////////////////////////////////////////////////////////////
    wire [31:0]offset;
    wire JALR;

    Decode inst1(
        .Instruction(Instruction_id),   // current instruction
        .MemtoReg(MemtoReg_id),         // use memory output as data to write into register
        .RegWrite(RegWrite_id),         // enable writing back to the register
        .MemWrite(MemWrite_id),         // write to memory
        .MemRead(MemRead_id),
        .ALUCode(ALUCode_id),               // ALU operation select
        .ALUSrcA(ALUSrcA_id),
        .ALUSrcB(ALUSrcB_id),
        .Jump(Jump),
        .JALR(JALR),
```

装 订 线

```
        .Imm(Imm_id),
        .offset(offset));

//////////////////////////////////////////////////////////////////////////////////
// Register
//////////////////////////////////////////////////////////////////////////////////

    Register inst2(
        .clk(clk),
        .rs1Addr(rs1Addr_id),
        .rs2Addr(rs2Addr_id),
        .WriteAddr(rdAddr_wb),
        .RegWrite(RegWrite_wb),
        .WriteData(RegWriteData_wb),
        .rs1Data(rs1Data_id),
        .rs2Data(rs2Data_id));

//////////////////////////////////////////////////////////////////////////////////
// Branch Test
//////////////////////////////////////////////////////////////////////////////////

    Branch inst3(
        .Instruction(Instruction_id),
        .rs1Data(rs1Data_id),
        .rs2Data(rs2Data_id),
        .Branch(Branch));

//////////////////////////////////////////////////////////////////////////////////
// Hazard Detector
//////////////////////////////////////////////////////////////////////////////////
    reg Stall;
    always @(*)
    begin
    if(((rdAddr_ex==rs1Addr_id) || (rdAddr_ex==rs2Addr_id)) && MemRead_ex)
    Stall=1'b1;
    else
    Stall=1'b0;
    end
    assign IFWrite=~Stall;
//////////////////////////////////////////////////////////////////////////////////
// Jump
//////////////////////////////////////////////////////////////////////////////////
```

装 订 线

```
    reg [31:0] JumpAddr;
    wire [31:0]JumpAddr1,JumpAddr2;

    adder_32bits inst4(
        .a(rs1Data_id),
        .b(offset),
        .ci(1'b0),
        .s(JumpAddr1),
        .co());

    adder_32bits inst5(
        .a(PC_id),
        .b(offset),
        .ci(1'b0),
        .s(JumpAddr2),
        .co());

    always @(*)
    begin
        case (JALR)
            1'b0:JumpAddr=JumpAddr2;
            1'b1:JumpAddr=JumpAddr1;
            default: JumpAddr=32'b0;
        endcase
    end

endmodule
```

装 订 线

3.1. Decode 模块代码

```
//*************************************************************************************
*
// Decode.v
//*************************************************************************************
*

module Decode(
    // Outputs
    MemtoReg, RegWrite, MemWrite,
MemRead,ALUCode,ALUSrcA,ALUSrcB,Jump,JALR,Imm,offset,
    // Inputs
    Instruction);
    input [31:0]    Instruction;    // current instruction
    output          MemtoReg;       // use memory output as data to write into register
    output          RegWrite;       // enable writing back to the register
```

```
    output              MemWrite;          // write to memory
    output              MemRead;
    output [3:0]        ALUCode;           // ALU operation select
    output              ALUSrcA;
    output [1:0]        ALUSrcB;
    output              Jump;
    output              JALR;
    output[31:0]        Imm,offset;

//**********************************************************************************
*
//   instruction type decode
//**********************************************************************************
*

    parameter   R_type_op=     7'b0110011;
    parameter   I_type_op=     7'b0010011;
    parameter   SB_type_op=    7'b1100011;
    parameter   LW_op=         7'b0000011;
    parameter   JALR_op=       7'b1100111;
    parameter   SW_op=         7'b0100011;
    parameter   LUI_op=        7'b0110111;
    parameter   AUIPC_op=      7'b0010111;
    parameter   JAL_op=        7'b1101111;
//
    parameter   ADD_funct3   =      3'b000 ;
    parameter   SUB_funct3   =      3'b000 ;
    parameter   SLL_funct3   =      3'b001 ;
    parameter   SLT_funct3   =      3'b010 ;
    parameter   SLTU_funct3 =       3'b011 ;
    parameter   XOR_funct3   =      3'b100 ;
    parameter   SRL_funct3   =      3'b101 ;
    parameter   SRA_funct3   =      3'b101 ;
    parameter   OR_funct3    =      3'b110 ;
    parameter   AND_funct3   =      3'b111;
//
    parameter   ADDI_funct3   =     3'b000 ;
    parameter   SLLI_funct3   =     3'b001 ;
    parameter   SLTI_funct3   =     3'b010 ;
    parameter   SLTIU_funct3 =      3'b011 ;
    parameter   XORI_funct3   =     3'b100 ;
    parameter   SRLI_funct3   =     3'b101 ;
    parameter   SRAI_funct3   =     3'b101 ;
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
    parameter   ORI_funct3    =       3'b101 ;
    parameter   ANDI_funct3   =       3'b111;
//
    parameter  alu_add=     4'b0000;
    parameter  alu_sub=     4'b0001;
    parameter  alu_lui=     4'b0010;
    parameter  alu_and=     4'b0011;
    parameter  alu_xor=     4'b0100;
    parameter  alu_or =     4'b0101;
    parameter       alu_sll=     4'b0110;
    parameter  alu_srl=     4'b0111;
    parameter  alu_sra=     4'b1000;
    parameter  alu_slt=     4'b1001;
    parameter  alu_sltu=    4'b1010;

//*************************************************************************
*
// instruction field
//*************************************************************************
*

    wire [6:0]      op;
    wire                funct6_7;
    wire [2:0]      funct3;
    assign op           = Instruction[6:0];
    assign funct6_7         = Instruction[30];
    assign funct3     = Instruction[14:12];
//
    wire R_type,I_type,SB_type,LW,JALR,SW,LUI,AUIPC,JAL;
//
    assign R_type=(op==R_type_op);
    assign I_type=(op==I_type_op);
    assign SB_type=(op==SB_type_op);
    assign LW=(op==LW_op);
    assign JALR=(op==JALR_op);
    assign SW=(op==SW_op);
    assign LUI=(op==LUI_op);
    assign AUIPC=(op==AUIPC_op);
    assign JAL=(op==JAL_op);
//
    assign MemtoReg=LW;
    assign MemRead=LW;
    assign MemWrite=SW;
```

装 订 线

```
    assign RegWrite=R_type||I_type||LW||JALR||LUI||AUIPC||JAL;

    assign ALUSrcA=JALR||JAL||AUIPC;
    assign ALUSrcB[1]=JAL||JALR;
    assign ALUSrcB[0]=~(R_type||JAL||JALR);

    assign Jump=JALR||JAL;

//*************************************************************************
*
//   ALUCode 定义
//*************************************************************************
*
///////////////根据 funct3,funct6_7 来决定 ALUCode//////////////////////////////////////////////////
    reg [3:0] ALUCode;

    always @(*)
    begin if ((R_type==1)&&(I_type==0)&&(LUI==0))
                begin if ((funct3==ADD_funct3)&&(funct6_7==0))
                    ALUCode=alu_add;
                else if ((funct3==SUB_funct3)&&(funct6_7==1))
                    ALUCode=alu_sub;
                else if ((funct3==SLL_funct3)&&(funct6_7==0))
                    ALUCode=alu_sll;
                else if ((funct3==SLT_funct3)&&(funct6_7==0))
                    ALUCode=alu_slt;
                else if ((funct3==SLTU_funct3)&&(funct6_7==0))
                    ALUCode=alu_sltu;
                else if ((funct3==XOR_funct3)&&(funct6_7==0))
                    ALUCode=alu_xor;
                else if ((funct3==SRL_funct3)&&(funct6_7==1))
                    ALUCode=alu_srl;
                else if ((funct3==SRA_funct3)&&(funct6_7==0))
                    ALUCode=alu_sra;
                else if ((funct3==OR_funct3)&&(funct6_7==0))
                    ALUCode=alu_or;
                else if ((funct3==AND_funct3)&&(funct6_7==0))
                    ALUCode=alu_and;
                end
            else if ((R_type==0)&&(I_type==1)&&(LUI==0))
                begin if ((funct3==ADDI_funct3)&&(funct6_7==0))
                    ALUCode=alu_add;
                else if ((funct3==SLLI_funct3)&&(funct6_7==0))
```

装 订 线

```
                    ALUCode=alu_sll;
                else if ((funct3==SLTI_funct3)&&(funct6_7==0))
                    ALUCode=alu_slt;
                else if ((funct3==SLTIU_funct3)&&(funct6_7==0))
                    ALUCode=alu_sltu;
                else if ((funct3==XORI_funct3)&&(funct6_7==0))
                    ALUCode=alu_xor;
            else if ((funct3==SRLI_funct3)&&(funct6_7==1))
                    ALUCode=alu_srl;
                else if ((funct3==SRAI_funct3)&&(funct6_7==0))
                    ALUCode=alu_sra;
                else if ((funct3==ORI_funct3)&&(funct6_7==0))
                    ALUCode=alu_or;
                else if ((funct3==ANDI_funct3)&&(funct6_7==0))
                    ALUCode=alu_and;
                end
            else if ((R_type==0)&&(I_type==0)&&(LUI==1))
                    ALUCode=alu_lui;
            else ALUCode=alu_add;
        end

//*************************************************************************
*
//  立即数生成
//*************************************************************************
*

        wire Shift;
        assign Shift=(funct3==1)||(funct3==5);

        reg [31:0] Imm,offset;

        always @(*)
        begin if ((I_type==1)&&(Shift==1))
                    Imm={26'b0,Instruction[25:20]};
            else if ((I_type==1)&&(Shift==0))
                    Imm={{20{Instruction[31]}},Instruction[31:20]};
            else if (LW==1)
                    Imm={{20{Instruction[31]}},Instruction[31:20]};
            else if (SW==1)
                    Imm={{20{Instruction[31]}},Instruction[31:25],Instruction[11:7]};
            else if (LUI==1)
                    Imm={Instruction[31:12],12'b0};
```

装 订 线

```
        else if (AUIPC==1)
            Imm={Instruction[31:12],12'b0};
        else
            Imm={32'b0};
    end

    always @(*)
    begin if (JALR==1)
            offset={{20{Instruction[31]}},Instruction[31:20]};
        else if (JAL==1)

offset={{11{Instruction[31]}},Instruction[31],Instruction[19:12],Instruction[20],Instruction[30:21
],1'b0};
        else if (SB_type==1)

offset={{19{Instruction[31]}},Instruction[31],Instruction[7],Instruction[30:25],Instruction[11:8],1
'b0};
        else
            offset={32'b0};
    end

endmodule
```

装

订

线

3.2. Register 模块代码

```
module Register(clk,rs1Addr,rs2Addr,WriteAddr,RegWrite,WriteData,rs1Data,rs2Data);
  input clk;
  input [4:0] rs1Addr;
  input [4:0] rs2Addr;
  input [4:0] WriteAddr;
  input RegWrite;
  input [31:0] WriteData;
  output   [31:0] rs1Data;
  output   [31:0] rs2Data;

  wire [31:0] ReadData1,ReadData2;

  RBWRegisters inst1(
    .clk(clk),
    .ReadRegister1(rs1Addr),
    .ReadRegister2(rs2Addr),
    .WriteRegister(WriteAddr),
    .RegWrite(RegWrite),
    .WriteData(WriteData),
    .ReadData1(ReadData1),
```

```
    .ReadData2(ReadData2));

  wire rs1Sel,rs2Sel;

  assign rs1Sel=RegWrite&&(WriteAddr!=0)&&(WriteAddr==rs1Addr);
  assign rs2Sel=RegWrite&&(WriteAddr!=0)&&(WriteAddr==rs2Addr);
//////////////////////////选择生成 rs1Data/////////////////////////////////////////
  mux32 inst2 (
          .out(rs1Data),
          .in0(ReadData1),
          .in1(WriteData),
          .addr(rs1Sel));
//////////////////////////选择生成 rs2Data/////////////////////////////////////////
  mux32 inst3 (
          .out(rs2Data),
          .in0(ReadData2),
          .in1(WriteData),
          .addr(rs2Sel));
endmodule
```

装

订

线

3.2.1. RBWRegisters 模块代码

```
module
RBWRegisters(clk,ReadRegister1,ReadRegister2,WriteRegister,RegWrite,WriteData,ReadData1,
ReadData2);
  input clk;
  input [4:0] ReadRegister1;
  input [4:0] ReadRegister2;
  input [4:0] WriteRegister;
  input RegWrite;
  input [31:0] WriteData;
  output [31:0] ReadData1;
  output [31:0] ReadData2;

  reg [31:0] regs[31:0];

  assign ReadData1=(ReadRegister1==5'b0)?32'b0:regs[ReadRegister1];
  assign ReadData2=(ReadRegister2==5'b0)?32'b0:regs[ReadRegister2];
  always @(posedge clk)
    if(RegWrite)
        regs[WriteRegister]<=WriteData;

endmodule
```

3.3. Branch 模块代码

```
module Branch(Instruction,rs1Data,rs2Data,Branch);
```

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
    input [31:0] Instruction;
    input [31:0] rs1Data,rs2Data;
    output reg Branch;

//
    parameter   beq_funct3   =       3'b000 ;
    parameter   bne_funct3   =       3'b001 ;
    parameter   blt_funct3   =       3'b100 ;
    parameter   bge_funct3   =       3'b101 ;
    parameter   bltu_funct3 =        3'b110 ;
    parameter   bgeu_funct3 =        3'b111 ;

    parameter   SB_type_op=    7'b1100011;
//
    wire isLT;
    wire isLTU;
    wire [31:0] sum;
    reg co;
//
    wire [6:0]      op;
    wire [2:0]      funct3;
    assign op          = Instruction[6:0];
    assign funct3      = Instruction[14:12];
//
    wire SB_type;
//
    assign SB_type=(op==SB_type_op);
//
    adder_32bits inst1(
        .a(rs1Data),
        .b((~rs2Data)),
        .ci(1'b1),
        .s(sum),
        .co());
    ////////////////////有无符号数比较大小结果//////////////////////////////////
    assign isLT=rs1Data[31]&&(~rs2Data[31])||(rs1Data[31]~^rs2Data[31])&&sum[31];
    assign isLTU=(~rs1Data[31])&&rs2Data[31]||(rs1Data[31]~^rs2Data[31])&&sum[31];

    always @(*)
    begin if(SB_type&&(funct3==beq_funct3))
            Branch=~(|sum[31]);
        else if(SB_type&&(funct3==bne_funct3))
            Branch=|sum[31];
```

装 订 线

```
        else if(SB_type&&(funct3==blt_funct3))
            Branch=isLT;
        else if(SB_type&&(funct3==bge_funct3))
            Branch=~isLT;
        else if(SB_type&&(funct3==bltu_funct3))
            Branch=isLTU;
        else if(SB_type&&(funct3==bgeu_funct3))
            Branch=~isLTU;
        else    Branch=1'b0;
    end

endmodule
```

4. EX 模块代码

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: zju
// Engineer: qmj
//////////////////////////////////////////////////////////////////////////////////
module EX(ALUCode_ex, ALUSrcA_ex, ALUSrcB_ex,Imm_ex, rs1Addr_ex, rs2Addr_ex,
rs1Data_ex,
            rs2Data_ex, PC_ex, RegWriteData_wb, ALUResult_mem,rdAddr_mem, rdAddr_wb,
            RegWrite_mem, RegWrite_wb, ALUResult_ex, MemWriteData_ex, ALU_A,
ALU_B);
    input [3:0] ALUCode_ex;
    input ALUSrcA_ex;
    input [1:0]ALUSrcB_ex;
    input [31:0] Imm_ex;
    input [4:0]    rs1Addr_ex;
    input [4:0]    rs2Addr_ex;
    input [31:0] rs1Data_ex;
    input [31:0] rs2Data_ex;
    input [31:0] PC_ex;
    input [31:0] RegWriteData_wb;
    input [31:0] ALUResult_mem;
    input [4:0]rdAddr_mem;
    input [4:0] rdAddr_wb;
    input RegWrite_mem;
    input RegWrite_wb;
    output [31:0] ALUResult_ex;
    output [31:0] MemWriteData_ex;
    output [31:0] ALU_A;
    output [31:0] ALU_B;
//////////////////////////////////////////////////////////////////////////////////
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
//ALU 子模块
////////////////////////////////////////////////////////////////////////////////
    ALU inst1(
        .ALUResult(ALUResult_ex),
        .ALUCode(ALUCode_ex),
        .A(ALU_A),
        .B(ALU_B));
////////////////////////////////////////////////////////////////////////////////
//Forward 子模块
////////////////////////////////////////////////////////////////////////////////
    wire [1:0] ForwardA,ForwardB;
    reg [31:0] A,B;

    assign
ForwardA[0]=RegWrite_wb&&(rdAddr_wb!=0)&&(rdAddr_mem!=rs1Addr_ex)&&(rdAddr_wb
==rs1Addr_ex);
    assign
ForwardA[1]=RegWrite_mem&&(rdAddr_mem!=0)&&(rdAddr_mem==rs1Addr_ex);
    assign
ForwardB[0]=RegWrite_wb&&(rdAddr_wb!=0)&&(rdAddr_mem!=rs2Addr_ex)&&(rdAddr_wb
==rs2Addr_ex);
    assign
ForwardB[1]=RegWrite_mem&&(rdAddr_mem!=0)&&(rdAddr_mem==rs2Addr_ex);
//////////////////////A 来源//////////////////////////////
    always @(*) begin
    case (ForwardA)
        2'b00:A=rs1Data_ex;
        2'b01:A=RegWriteData_wb;
        2'b10:A=ALUResult_mem;
        default: A=32'b0;
    endcase
    end
//////////////////B 来源//////////////////////////////
    always @(*) begin
    case (ForwardB)
        2'b00:B=rs2Data_ex;
        2'b01:B=RegWriteData_wb;
        2'b10:B=ALUResult_mem;
        default: B=32'b0;
    endcase
    end

    assign MemWriteData_ex=B;
```

装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
////////////////////////////////////////////////
//////////////////ALU_A 来源//////////////////////
    reg [31:0] ALU_A,ALU_B;
    always @(*) begin
    case (ALUSrcA_ex)
        1'b0:ALU_A=A;
        1'b1:ALU_A=PC_ex;
        default: ALU_A=32'b0;
    endcase
    end
//////////////////ALU_B 来源//////////////////////
    always @(*) begin
    case (ALUSrcB_ex)
        2'b00:ALU_B=B;
        2'b01:ALU_B=Imm_ex;
        2'b10:ALU_B=32'b100;
        default: ALU_B=32'b0;
    endcase
    end

endmodule
```

装 订 线

4.1. ALU 模块代码

```
//*************************************************************************
*
// RISC V verilog model
// ALU.v
//*************************************************************************
*

module ALU (
    // Outputs
        ALUResult,
    // Inputs
        ALUCode, A, B);

    input [3:0]     ALUCode;                    // Operation select
    input [31:0]    A, B;
    output [31:0]   ALUResult;

// Decoded ALU operation select (ALUsel) signals
    parameter   alu_add=   4'b0000;
    parameter   alu_sub=   4'b0001;
    parameter   alu_lui=   4'b0010;
```

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
    parameter    alu_and=    4'b0011;
    parameter    alu_xor=    4'b0100;
    parameter    alu_or =    4'b0101;
    parameter    alu_sll=    4'b0110;
    parameter    alu_srl=    4'b0111;
    parameter    alu_sra=    4'b1000;
    parameter    alu_slt=    4'b1001;
    parameter    alu_sltu= 4'b1010;

    reg signed [31:0] A_reg;
    always @(*)
            begin A_reg=A;
            end

    wire Binvert;
    wire [31:0] sum;
    wire [31:0] B1;
    assign Binvert=~(ALUCode==0);
    assign B1=B^{32{Binvert}};

    adder_32bits inst1(
            .a(A),
            .b(B1),
            .ci(Binvert),
            .s(sum),
            .co());

    reg [31:0] ALUResult;

    always @(*)    begin
            case (ALUCode)
                    alu_add : ALUResult=sum;
                    alu_sub : ALUResult=sum;
                    alu_lui : ALUResult=B;
                    alu_and : ALUResult=A&B;
                    alu_xor : ALUResult=A^B;
                    alu_or    : ALUResult=A|B;
                    alu_sll : ALUResult=A<<B;
                    alu_srl : ALUResult=A>>B;
                    alu_sra : ALUResult=A_reg>>>B;
                    alu_slt : ALUResult=A[31]&&(~B[31])||(A[31]~^B[31])&&sum[31];
                    alu_sltu: ALUResult=(~A[31])&&B[31]||(A[31]~^B[31])&&sum[31];
                    default:ALUResult=32'b0;
```

装 订 线

```
            endcase
            end
endmodule
```

5. dffre 模块代码

```
module dffre(d,en,r,clk,q);
        parameter n=1;
        input clk,en,r;
        input[n-1:0] d;
        output[n-1:0] q;
        reg [n-1:0] q;
        always @(posedge clk)
                if(r)q={n{1'b0}};
                else if (en)q=d;
                else q=q;
endmodule
```

6. adder_32bits 模块代码

```
module adder_32bits(a,b,ci,s,co);
   input [31:0]a,b; //输入两个 32 位的加数
   input ci;//输入最低位进位
   output [31:0]s;//输出两个 32 位加数的和
   output co;//输出最高位进位
   wire c3,c7,c11,c15,c19,c23,c27,c31;

   adder_4bits inst0(//最低 4 位超前进位加法器
      .a(a[3:0]),
      .b(b[3:0]),
      .ci(ci),
      .s(s[3:0]),
      .co(c3));
   adder_4bitsx2 inst1(//由两个 4 位超前进位加法器经门电路、MUX 组合成的加法结构块
      .a(a[7:4]),
      .b(b[7:4]),
      .ci(c3),
      .s(s[7:4]),
      .co(c7));
   adder_4bitsx2 inst2(
      .a(a[11:8]),
      .b(b[11:8]),
      .ci(c7),
      .s(s[11:8]),
      .co(c11));
   adder_4bitsx2 inst3(
      .a(a[15:12]),
```

装 订 线

```
        .b(b[15:12]),
        .ci(c11),
        .s(s[15:12]),
        .co(c15));
    adder_4bitsx2 inst4(
        .a(a[19:16]),
        .b(b[19:16]),
        .ci(c15),
        .s(s[19:16]),
        .co(c19));
    adder_4bitsx2 inst5(
        .a(a[23:20]),
        .b(b[23:20]),
        .ci(c19),
        .s(s[23:20]),
        .co(c23));
    adder_4bitsx2 inst6(
        .a(a[27:24]),
        .b(b[27:24]),
        .ci(c23),
        .s(s[27:24]),
        .co(c27));
    adder_4bitsx2 inst7(
        .a(a[31:28]),
        .b(b[31:28]),
        .ci(c27),
        .s(s[31:28]),
        .co(c31));
    assign co=c31;
endmodule
//////////////////////////////////////////////////////////////////
module adder_4bits(a,b,ci,s,co);
    input [3:0] a,b;
    input ci;
    output co;
    output [3:0] s;
    wire [3:0] p,g;
    wire c0,c1,c2;

    assign p=a^b;
    assign g=a&b;

    adder adder_inst1(
```

装 订 线

```
    .pi(p[0]),
    .gi(g[0]),
    .cin(ci),
    .cout(c0),
    .s(s[0])
  );
  adder adder_inst2(
    .pi(p[1]),
    .gi(g[1]),
    .cin(c0),
    .cout(c1),
    .s(s[1])
  );
  adder adder_inst3(
    .pi(p[2]),
    .gi(g[2]),
    .cin(c1),
    .cout(c2),
    .s(s[2])
  );
  adder adder_inst4(
    .pi(p[3]),
    .gi(g[3]),
    .cin(c2),
    .cout(co),
    .s(s[3])
  );
endmodule
//////////////////////////////////////////////////////////////////////////////
module adder_4bitsx2(a,b,ci,s,co);//由两个 4 位超前进位加法器经门电路、MUX 组合成的加法结构块
  input [3:0]a,b;
  input ci;
  output [3:0]s;
  output co;
  wire c1,c0;
  wire [3:0]s1,s0;

   adder_4bits inst1(//4 位超前进位加法器
    .a(a),
    .b(b),
    .ci(1'b1),
    .s(s1),
```

装 订 线

```
      .co(c1));
    adder_4bits inst2(
      .a(a),
      .b(b),
      .ci(1'b0),
      .s(s0),
      .co(c0));
    assign co=(ci&c1)|c0;//输出进位
    mux inst3(//2 to 1 MUX
    .out(s),
    .in0(s0),
    .in1(s1),
    .addr(ci));
endmodule
//////////////////////////////////////////
module adder(pi,gi,cin,cout,s);
  input pi,gi;
  input cin;
  output cout,s;
  assign s=(pi&(~gi))^cin;
  assign cout=gi|(pi&cin);
endmodule
```

装 订 线

7. mux 模块代码

```
module mux(out,in0,in1,addr);//2to1 MUX
  output[3:0] out;
  input[3:0] in0,in1;
  input addr;
  reg[3:0] out;

  always @(in0 or in1 or addr)    //地址为 1'b0 时，输出 in0,地址为 1'b1 时，输出 in1
    begin
      if(addr)out=in1;
      else out=in0;
    end
endmodule
```

8. mux32 模块代码

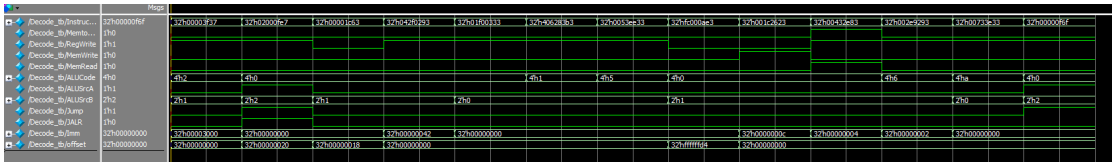
```
module mux32(out,in0,in1,addr);//2to1 MUX
  output[31:0] out;
  input[31:0] in0,in1;
  input addr;
  reg[31:0] out;

  always @(in0 or in1 or addr)    //地址为 1'b0 时，输出 in0,地址为 1'b1 时，输出 in1
```

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
    begin
      if(addr)out=in1;
      else out=in0;
    end
endmodule
```

五、仿真过程分析

1. Decode 模块仿真



测试代码如下：

Instruction = 32'h00003f37;//lui　X30, 0x3000

需要回写，因而 RegWrite=1，Imm=00003000，offset 无意义

Instruction = 32'h02000fe7;//jalr X31, later(X0)

跳转指令，Jump=1，需要回写，RegWrite=1，imm 无意义，offset=00000020

Instruction = 32'h00001c63;//bne　X0, X0, end

不需要回写，RegWrite=0, imm 无意义，offset=00000018

Instruction = 32'h042f0293;//addi X5, X30, 42

需要回写，RegWrite=1, imm=00000042，offset 无意义

Instruction = 32'h01f00333;//add　X6, X0, X31

需要回写，RegWrite=1, imm 无意义，offset 无意义

Instruction　= 32'h406283b3;//sub　X7, X5, X6

需要回写，RegWrite=1, imm 无意义，offset 无意义

Instruction = 32'h0053ee33;//or　　X28, X7, X5

需要回写，RegWrite=1, imm 无意义，offset 无意义

Instruction = 32'hfc000ae3;//beq X0, X0, earlier

不需要回写，RegWrite=0, imm 无意义，offset=ffffffd4

Instruction = 32'h001c2623;//sw　X28, 0C(X0)

不需要回写，RegWrite=0, imm=0000000c，offset 无意义, sw 执行对内存的写操作，所以 MemWrite=1, imm=0000000c, offset 无意义

Instruction = 32'h00432e83;//lw X29, 04(X6)

lw 执行对于内存的读操作，所以 MemRead=1, MemtoReg=1, 需要回写，RegWrite=1，imm=00000004, offset 无意义

Instruction = 32'h002e9293;//sll X5, X29,2

需要回写，RegWrite=1，imm=00000002, offset 无意义

Instruction = 32'h00733e33;//sltu X28, X6,X7

需要回写，RegWrite=1，imm 无意义, offset 无意义

Instruction = 32'h0000f6f;//jal　X31, done

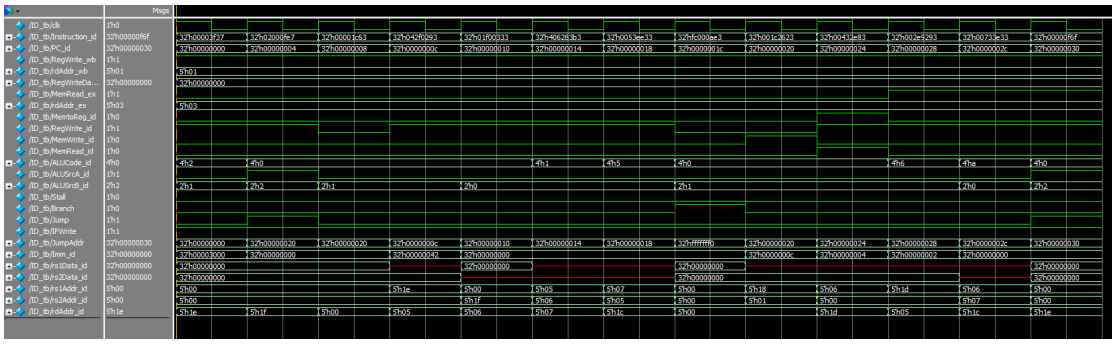
跳转指令，所以 Jump=1，需要回写，RegWrite=1，imm 无意义, offset 无意义

综合以上所有的指令分析，可以得知，该子模块运行正确。

2. ID 模块仿真

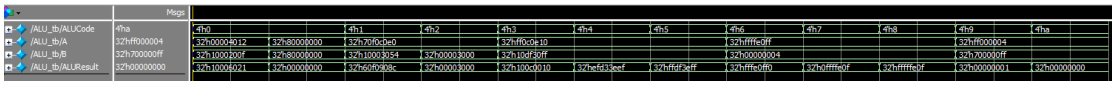
装 订 线

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212



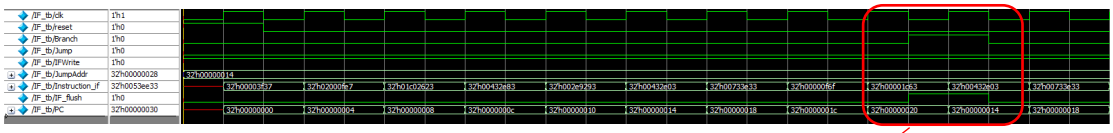
由于 Decode 模块已经测试过，因而只需关注其他的输出量，可以看到，Branch、Stall、IFWrite 输出逻辑关系正确，根据指令直接解析出的 rs1Addr_id、rs2Addr_id，rdAddr_id 显示正确。此外由于 Register 部分比较简单，因而采取直接检查语法错误的方式。综合以上分析，ID 模块工作正常。

3. ALU 模块仿真

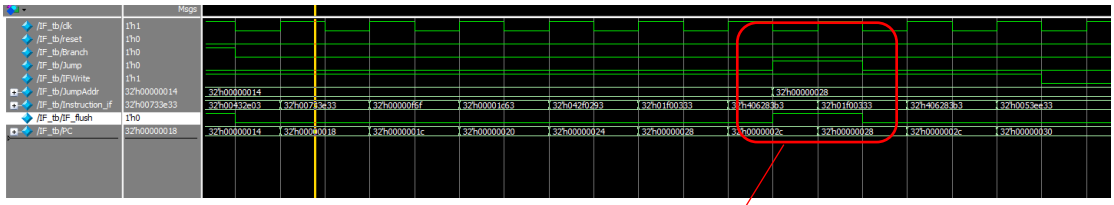


对照 ALU 功能表，仿真结果正确。说明 ALU 子模块设计符合要求。

4. IF 模块仿真



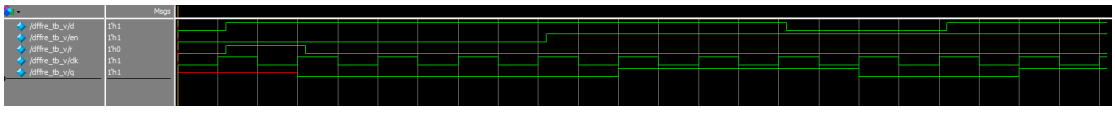
Branch=1，发生跳转，下一 PC 值为 JumpAddr 的值，即 00000014



Jump=1，发生跳转，下一 PC 值为 JumpAddr 的值，即 00000028

总体来看，PC 值跳转正常，读取指令正常，因而，IF 模块工作正常。

5. dffre 模块仿真



当 r=1 时 q 重置为 0，当 en=0 时保持。证明该模块工作正常。

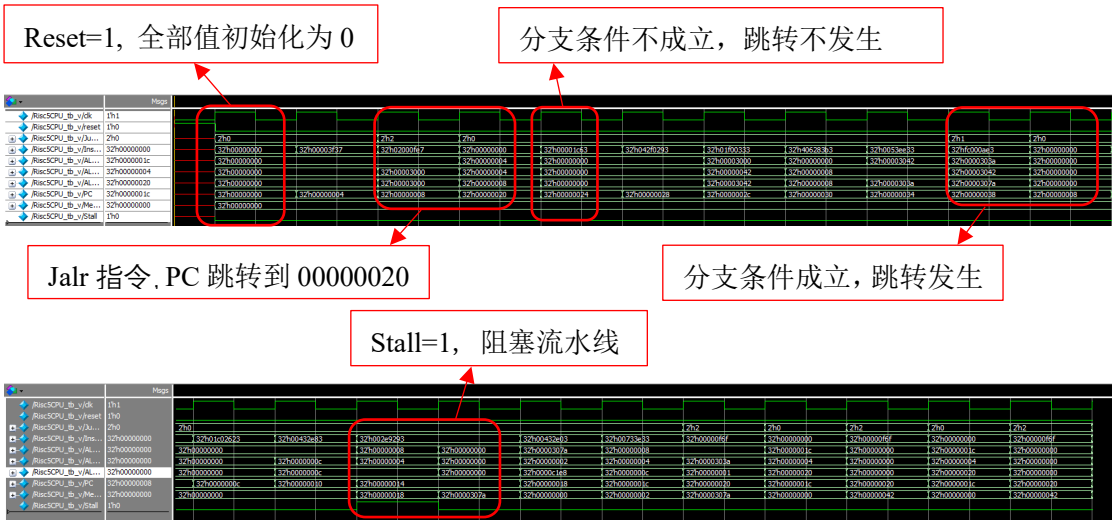
6. 顶层功能仿真

顶层测试程序如下：

```
6'd0:    dout=32'h0000_3f37 ;//          lui X30,0x3000
6'd1:    dout=32'h0200_0fE7 ;//          jalr X31 later(X0)
```

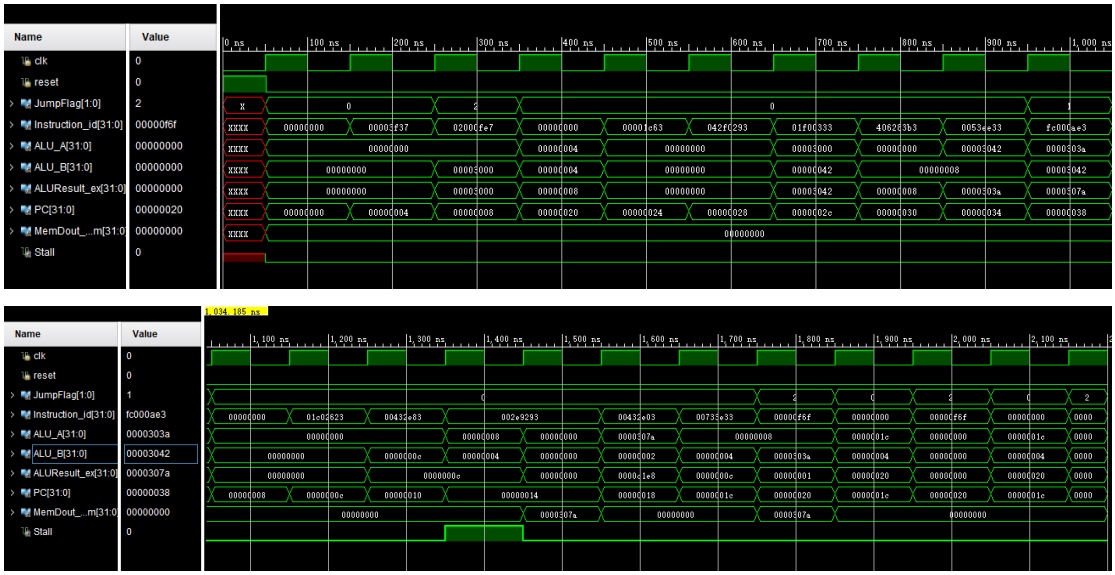
实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

```
6'd2:    dout=32'h01c0_2623 ;// earlier: sw   X28, 0C(X0)
6'd3:    dout=32'h0043_2e83 ;//               lw   X29, 4(X6)
6'd4:    dout=32'h002e_9293 ;//               sll  X5, X29, 2
6'd5:    dout=32'h0043_2e03 ;//               lw   X28, 4(X6)
6'd6:    dout=32'h0073_3e33 ;//               sltu X28,X6,X7
6'd7:    dout=32'h0000_0f6f ;//  done:        jal X31,done
6'd8:    dout=32'h0000_1c63 ;//  later:       bne X0, X0, end    // 分支条件不成立
6'd9:    dout=32'h042f_0293;//                addi X5, X30, 42
6'd10:   dout=32'h01f0_0333 ;//               add  X6, X0, X31
6'd11:   dout=32'h4062_83b3 ;//               sub X7, X5, X6
6'd12:   dout=32'h0053_ee33 ;//               or   X28, X7, X5
6'd13:   dout=32'hfc00_0ae3 ;//               beq X0, X0, earlier
6'd14: dout=32'h00000000 ;// end:       nop
default:dout=32'h00000000 ;//nop
```



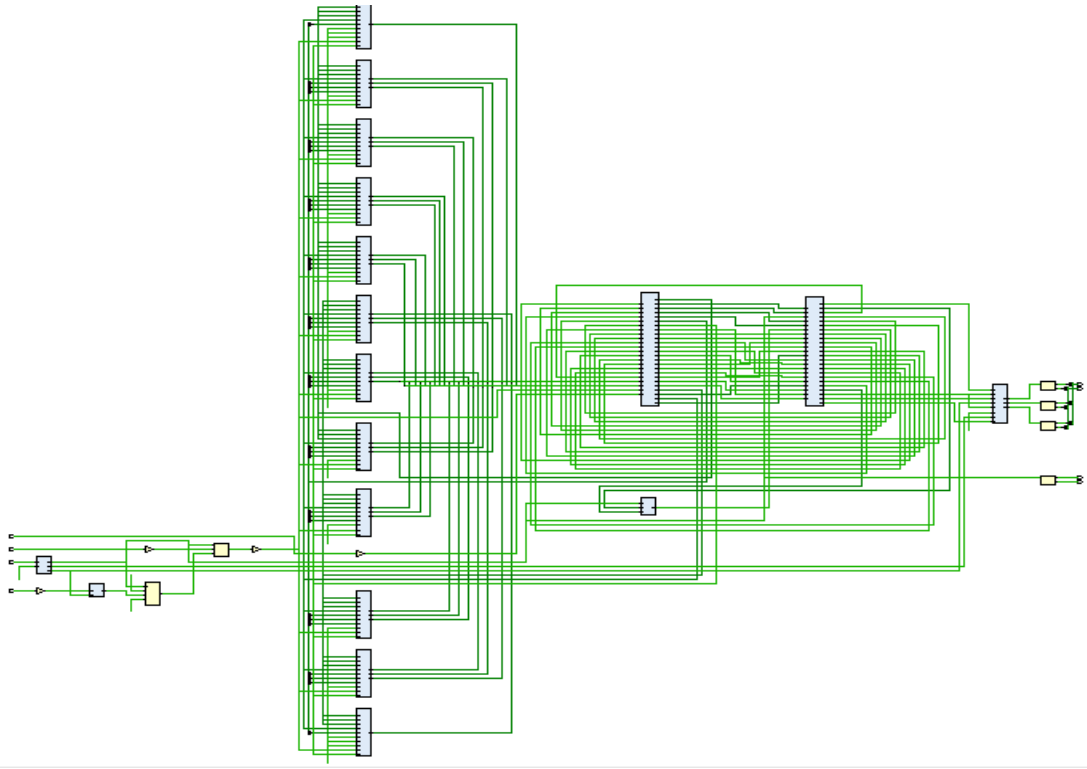
综上，整个流水线 CPU 工作正常。

7. 顶层时序仿真



采用 Vivado 进行补充时序仿真的验证，得到结果完全正确。

8、综合实现结果



装 订 线

六、心得与体会

本次实验过程总体上进展比较顺利。通过这次实验我对于 RISC-V 指令集及流水线 CPU 设计的认识均有了很大程度的加深。实验课的练习同时强化了我对于理论知识的理解，自身获益匪浅。

老师给予的实验指导资料中已经很明确的给出了各个模块中的工作原理与信号之间的逻辑关系，还有具体的电路图，在这些资料的指导下，凭借耐心，一步步从小模块到顶层模块，逐步调用，进而完成整项作业。

实验过程中，由于连接信号众多，信号名称相近，很容易由于录入粗心而产生错误，这也是本次实验最主要的错误来源。

此外，我认为本次实验的难点在于对流水线 CPU 多种工作状态的整体把握，以及顶层仿真中的调试 Debug 工作。流水线 CPU 速度更快，多条指令同时运行，这就要求我们对不同指令的运行以及数据的传递都要有一定的认识和掌握。在理论课程中已经对流水线 CPU 的工作方式有了一定的了解学习，通过实验可以更直观更透彻的学习其具体流程和实现方式。

由于各子模块的逻辑关系清晰、结构清晰、原理明确，仿真中出现的错误很容易纠正，但在顶层模块调试过程中存在各级数据转发，而且同一指令在各级运行的时间不同，因而错误的排查上具有一定困难。

整个实验中我出现的错误及耗费的时间也主要在顶层仿真中。出现错误时我一般会先观察错误波形所对应的输入波形是否正确，若正确再将其中子模块的信号添加进波形里，进行观察。如果输入信号全部正常，则问题在于错误波形所在模块，仔细检查模块的各个输入输出信号之间的逻辑关系是否正确，再具体排查一遍代码中是否存在语法或者其他小问题，基本可以发现错误所在。我在顶层仿真时出现 Mem 级读取不正确的问题，经过逐层查找，发现有一 wire 变量未定义，造成不定态，因而错误。

实验名称：基于 RV32I 指令集的 RISC-V 微处理器设计 姓名：李依林 学号：3170101212

此外，这次实验也使我重新回顾了 IP 内核的生成方式，和仿真方式。

回顾整个实验过程，最重要的是严谨细致的态度以及足够的耐心。只要每个子模块都再细心一点，最后实验的调试就能更轻松一点。

实验中的不足之处在于对调试方法的掌握还是不够数量、对于软件的技巧性掌握不够到位，很多功能仍未学习到，一些错误提示也不理解，还需要进一步学习。